FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Generic and Parameterizable Service for Remote Configuration of Mobile Phones Using Near Field Communication

Carlos Tiago Rocha Babo

Mestrado Integrado em Engenharia Informática e Computação

Advisor: Prof. Hugo Sereno Ferreira (PhD)

July 15, 2013

Generic and Parameterizable Service for Remote Configuration of Mobile Phones Using Near Field Communication

Carlos Tiago Rocha Babo

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Ademar Manuel Teixeira de Aguiar (PhD)

External Examiner: Helena Cristina Coutinho Duarte Rodrigues (PhD)

Supervisor: Hugo José Sereno Lopes Ferreira (PhD)

July 15, 2013

Abstract

Mobile services have increased both in number and complexity in the past few years. This means that in order to get the most out of these services, less experienced users will have a hard time configuring them by hand. To address this issue, we must find new and innovative solutions to assist the user in this process. Furthermore, we live in a society of the immediate: everyone wants access to resources to be fast, simple and secure. It is also known that most of the users are laymen when referring to advanced configuration of mobile phone, resulting in some inertia in the use of applications and functionalities.

The survey of the State-of-the-Art in this field shows that currently there is no complete solution to this problem. However, the related work shows that some of the problems are already being addressed and that with Near Field Communication it is possible to develop new solutions, successfully reducing user intervention.

Near Field Communication (NFC) provides an unique opportunity to introduce new business paradigms in terms of interaction and ease of use. This technology allows short-range communications between NFC-enabled devices by simply holding them near each other, typically with minimal user intervention.

What mechanisms can one use to minimise an user's effort to configure his mobile device? What technologies are available that cope with this problem? How can one leverage them? What mechanisms can be used to ensure their security? How could service providers take advantage of such service? This dissertation provides an answer for these questions by proposing an architecture that defines a new logic layer that can be used by vendors and developers to configure applications and the mobile phone itself. It supports multiple location-based service configurations, with or without authentication, for different users. The proof-of-concept prototypes show promising results, allowing multiple services to be configured simultaneously and effortlessly.

Resumo

Os serviços nos nossos dispositivos móveis têm aumentado em número e complexidade nos últimos anos. Utilizadores menos experientes sentem dificuldade em tirar total partido destes serviços. De forma a atenuar este problema, é necessário encontrar novas e inovadoras formas que permitam assistir o utilizador no processo de configuração. Para além disso, vivemos numa sociedade do imediato: as pessoas querem que o acesso aos recursos seja rápido, simples e seguro. É também sabido que grande parte dos utilizadores são leigos no que diz respeito à utilização de funcionalidades avançadas dos dispositivos móveis, o que resulta em alguma inércia no uso de certas aplicações e funcionalidades.

O levantamento do estado da arte mostra que atualmente não há nenhuma solução que resolva totalmente o problema. No entanto, o trabalho relacionado demonstra alguns projetos que tentam resolver parte do problema e que com Near Field Communication é possível desenvolver novas soluções, tipicamente com reduzida intervenção do utilizador.

A tecnologia NFC apresenta uma oportunidade única de introduzir novos paradigmas de negócio em termos de interação e facilidade de uso. Esta tecnologia permite comunicações a curta distância entre dispositivos NFC, segurando-os próximo um do outro, e geralmente com intervenção mínima do utilizador.

Que mecanismos podem ser usados para diminuir o esforço do utilizador na configuração do seu dispositivo móvel? Que tecnologias podem ajudar a resolver este problema? Como se pode tirar partido delas? Que mecanismos oferecem segurança neste processo? De que forma os provedores de serviços podem tirar vantagem deste sistema? Estas questões são resolvidas com a proposta de uma arquitetura que define uma nova camada lógica que pode ser usada por terceiros de forma a providenciar uma nova forma de configurar aplicações e o telemóvel em si. Para além disso, esta arquitetura suporta múltiplas configurações de serviços, baseando-se na localização, com ou sem autenticação e para diferentes utilizadores. Os protótipos desenvolvidos mostram resultados promissores, permitindo que diversos serviços sejam configurados em simultâneo e sem esforço.

Acknowledgements

First of all, I thank my supervisors, Luís Carvalho and Hugo Ferreira, for their guidance throughout my dissertation.

I would also like to thank Fraunhofer AICOS for an awesome work environment and for granting me a research scholarship.

Finally, I want to thank all my family and friends that contributed to my path along the past five years.

Tiago Babo

"Software and cathedrals are much the same.
First we build them, then we pray."

Sam Redwine

Contents

1 Introduction **1**
- 1.1 Motivation and Objectives . . . 2
- 1.2 Real World Scenarios . . . 4
  - 1.2.1 Airport . . . 4
  - 1.2.2 Tourism center . . . 4
  - 1.2.3 Train station . . . 4
- 1.3 Report structure . . . 5

2 Wireless Technologies **7**
- 2.1 Near Field Communication . . . 7
  - 2.1.1 Specification . . . 7
  - 2.1.2 Security . . . 9
  - 2.1.3 NFC Data Exchange Format . . . 11
- 2.2 Other wireless technologies . . . 11
  - 2.2.1 Wi-Fi Direct . . . 12
  - 2.2.2 Bluetooth . . . 12
  - 2.2.3 IrDA . . . 13
  - 2.2.4 Comparison with NFC . . . 13
- 2.3 Summary . . . 14

3 Related Work **15**
- 3.1 Remote Device Management Protocols . . . 15
  - 3.1.1 Open Mobile Alliance Device Management . . . 15
  - 3.1.2 Broadband Forum - CPE WAN Management . . . 17
  - 3.1.3 Discussion . . . 17
- 3.2 Mobile Device Configuration using NFC . . . 18
  - 3.2.1 InstaWifi . . . 18
  - 3.2.2 NFC Task Launcher . . . 19
  - 3.2.3 Means for provisioning and managing mobile device configuration over a near-field communication link . . . 19
  - 3.2.4 Secure Hotspot Authentication through a Near Field Communication Side-Channel . . . 20
  - 3.2.5 Discussion . . . 21
- 3.3 Remote Configuration of Devices using a Mobile Phone . . . 21
  - 3.3.1 AppNearMe . . . 21
  - 3.3.2 Discussion . . . 21
- 3.4 Pairing of Devices using NFC tags . . . 22
  - 3.4.1 Zero-Configuration of Pervasive Healthcare Sensor Networks . . . 22

3.4.2 Design and Evaluation of a Telemonitoring Concept Based on NFC-Enabled Mobile Phones and Sensor Devices . . . 23
3.4.3 Discussion . . . 23
3.5 Conclusions . . . 23

4 System Requirements Specification **25**
4.1 Use Case Model . . . 25
4.1.1 Actors . . . 25
4.1.2 Service Provider . . . 26
4.1.3 Device . . . 26
4.2 Non-Functional Requirements . . . 27
4.2.1 Extensibility . . . 28
4.2.2 Usability . . . 28
4.2.3 Security . . . 28
4.2.4 Timeliness . . . 29
4.3 Summary . . . 29

5 Architecture and Design of the Prototype **31**
5.1 Main Server . . . 31
5.2 Local Server . . . 33
5.3 Client . . . 34
5.3.1 Subscribing a Configuration . . . 35
5.4 Data Exchange Protocol . . . 35
5.5 Encryption Protocol . . . 35
5.6 Summary . . . 36

6 Implementation Details **39**
6.1 Main server . . . 39
6.1.1 Overview . . . 40
6.1.2 Handling configuration requests . . . 41
6.1.3 Interfaces . . . 42
6.2 Client . . . 43
6.2.1 Android . . . 43
6.2.2 Generic client . . . 45
6.2.3 Configuration Process . . . 46
6.2.4 Handling Configuration Requests through NFC . . . 47
6.2.5 Interfaces . . . 48
6.3 Local Server . . . 48
6.3.1 Generic Local Server . . . 48
6.3.2 ACR122U NFC Reader . . . 49
6.3.3 Android Custom Rom . . . 55
6.4 ULF-MC Helmholtz Coils . . . 57
6.4.1 Using the Library to Send Configurations . . . 58
6.5 Test Applications . . . 59
6.5.1 Wifi Configurator . . . 59
6.5.2 RSS Reader . . . 59
6.6 Functional Tests . . . 59
6.7 Summary . . . 60

7 Conclusion and Future Work **63**

A Main Server Models **65**

B Main Server Interfaces **67**

References **69**

List of Figures

2.1 Man-in-the-middle setup scenario. 11
2.2 General view of an NDEF message [For06]. 12

3.1 Overview of OMA DM two phases protocol - setup and management phase. . . . 17
3.2 Example of a configuration process using the Broadband Forum - CPE WAN Management protocol. 18
3.3 InstaWifi screen shots showing the interface for writing a Wi-Fi configuration in an NFC tag or sharing it via an auto-generated QR Code [Che13]. 19
3.4 NFC Task Launcher activities for choosing a pre-defined task and for writing the selected set of configurations in an NFC tag [Tag13b]. 20
3.5 AppNearMe Developer activities for managing the current device configurations and for controlling an external lamp state and color through NFC. [Tag13b]. . . . 22

4.1 NFC Service Provider's use case diagram. 27
4.2 Device's use case diagram. 28

5.1 Architecture overview diagram. It can be decomposed in three modules: the Main Server, where all configurations are saved and managed; the Local Server, responsible for connecting the Client and the Main Server; the Client, a mobile phone application that provides configurations to third party applications. 32
5.2 Class diagram of the main server. Each configuration can have multiple values and users associated. Each user can be associated to multiple configuration. A user is characterized by an email, which is mandatory and a password, which is optional. 32
5.3 Sequence diagram of a generic configuration process. 34
5.4 Main cryptographic operations between a Client and a Local Server. 37

6.1 Final schema of the the main server's database. Each configuration can have multiple values and users associated. Each user can be associated to multiple configuration. 40
6.2 Graphical representation of the algorithm responsible for selecting valid configurations for a specific request. 42
6.3 Tag dispatch system [Goo13d]. 45
6.4 Client application activities. The left one represents the launch activity, and on the right one a user can change his authentication data. 49
6.5 ACS ACR122U NFC Reader, a contactless smart card reader[1]. 50

B.1 Main page interface example of the Main Server web application. 67
B.2 Configuration page interface of the Main Server web application. 68

LIST OF FIGURES

List of Tables

2.1 Communication configurations of NFC [Int08]. 8
2.2 Possible role combinations between Active/Passive and Initiator/Target. 8
2.3 NFC Forum Device communication links [Gal11]. 9
2.4 Comparison between NFC, Bluetooth, IrDA and Wi-Fi Direct [Ele06, Kum10, Hop09]. 13

3.1 Related work summary. 16

6.1 The Main Server provides different value types. This table shows an example for each one. 41
6.2 Command to configure the PN532 as target. 51
6.3 Format of the connect APDU. 52
6.4 Format of the information APDU. 52
6.5 SNEP put request message format. 52
6.6 NDEF record layout. 53
6.7 Format of the connection complete APDU. 53
6.8 Format of the receive ready APDU. 54
6.9 Format of the disconnect mode APDU. 54
6.10 Format of a wireless configuration usign the ULF-MC prototype. 58
6.11 Functional test 1. 60
6.12 Functional test 2. 60
6.13 Functional test 3. 61
6.14 Functional test 4. 62
6.15 Functional test 5. 62

Abbreviations

3DES	Triple Data Encryption Algorithm
AES	Advanced Encryption Standard
ACS	Auto-Configuration Server
AP	Access Point
API	Application Programming Interface
BSIG	Bluetooth Special Interest Group
CPE	Customer Premises Equipment
DSAP	Destination Service Access Point
EAP	Extensible Authentication Protocol
ECMA	European Computer Manufacturers Association
ETSI	European Telecommunications Standards Institute
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IrDa	Infrared Data Association
ISO	International Organization for Standardization
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
LLCP	Logical Link Control Protocol
MIME	Multipurpose Internet Mail Extensions
MSc	Master of Science
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NFCIP	Near Field Communication Interface and Protocol
OMA	Open Mobile Alliance
PIN	Personal Identification Number
QR	Quick Response
REST	Representational State Transfer
RF	Radio Frequency
RFID	Radio-Frequency Identification
RSS	Rich Site Summary
SDK	Software Development Kit
SNEP	Simple NDEF Exchange Protocol
SOAP	Simple Object Access Protocol
SSAP	Source Service Access Point
SSID	Service Set IDentifier

SSL	Secure Sockets Layer
TLS	Transport Layer Security
ULF-MC	Ultra Low Frequency Magnetic Field Communication
URI	Uniform Resource Identifier
WAN	Wide Area Network
WAP	Wireless Application Protocol
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access II
WPS	Wifi Protected Setup
XML	Extensible Markup Language

Chapter 1

Introduction

The International Telecommunication Union estimates that at the end of 2013 there will be 6.8 billion mobile-cellular subscriptions [Uni13]. Mobile-cellular penetration rates stand at 96% globally; 128% in developed countries; and 89% in developing countries. We have reached a moment where there are almost as many mobile-cellular subscriptions as people in the world. We also know that most of the users carry their device with or near them wherever they go [PKH+06]. These devices are powerful, personal and considered to be an extension of our digital selves [DL12]. The previsions estimate that this tendency will not stop and users will ultimately want a device that gathers functionalities of today's everyday objects, just like a swiss army knife [ABPW07, DL12].

Furthermore, we are facing an era of great opportunities in terms of innovative content and services. In particular, systems will tend to be more complex and difficult to manage and configure, thus being necessary to find new and easy ways of solving these problems. Near Field Communication (NFC) provides an unique opportunity to introduce new business paradigms in terms of interaction and ease of use [MP11]. This technology allows short-range communications between NFC-enabled devices by simply holding them near each other, typically with minimal user intervention [EA12, Kum10].

The first attempt to bring NFC to mobile phones was by Nokia, in 2006 [Kum10, Ker11]. Due to the lack of support by the developers and security problems, it wasn't successful. In 2011, Google announced the Samsung Nexus with NFC, bringing the technology to the spotlight again. Handset vendors have since shown their support to the technology, releasing more than 40 NFC-enabled handsets [Dav12].

Nowadays, NFC is mainly used has a contactless payment system, replacing the traditional credit card and electronic smartcards [Fis09, Cru11]. There are already different systems that allow the users to store bank information in mobile devices and use it as a payment method - Google Wallet [Goo12] and MasterCard PayPass [Mas12], for example. Moreover, different trials have been conducted in Austria [Tim07], Germany [Tim10], India [Upa12], Italy [Cla09], New Zealand[Cla12] and other countries with NFC-based ticketing systems for public transports.

In the next few years, the use of NFC in mobile phones foreshadows a new revenue stream to vendors, infrastructures operators and payment companies [Fis09]. Remote configuration through NFC is one of the areas that it is starting to grow. Projects are under development and others proved the viability of this area [Wag08].

1.1 Motivation and Objectives

Mobile services have increased both in number and complexity in the past few years. This means that in order to get the most out of these services, less experienced users will have a hard time configuring them by hand. It is also known that most of the users are laymen when referring to advanced configuration of mobile phone, resulting in some inertia in the use of applications and functionalities. To address this issue, we must find innovative solutions to assist the user in this process.

What mechanisms can one use to minimise an user's effort to configure his mobile device for certain tasks, e.g., Internet Access? What technologies are available that cope with this problem? How can one leverage them? What mechanisms can be used to ensure their security? How could service providers take advantage of such service? This dissertation aims to provide an answer for these questions, proposing an architecture for a remote configuration service for mobile phones through Near Field Communication, with minimal user intervention and technical knowledge. Unlike Wi-Fi or Bluetooth, there is no need of an initial setup time [Kum10] - normally the user only needs to activate the functionality.

Geven et al. conducted a study to better understand the possibilities of NFC in real-world interaction [GMS08]. Participants were asked to rate NFC door access and payment in comparison to the traditional methods they used:

- *"Access: 58 percent rate NFC as faster than their separate access card, 92 percent as more comfortable, 73 percent as more secure, 82 percent as better, 89 percent as more user friendly, 97 percent as 'cooler'.*
- *Payment: 89 percent rate NFC as faster than payment in cash, 95 percent as more comfortable, 40 percent as more secure, 90 percent as better, 90 percent as more user friendly, and 95 percent as 'cooler'."*

The study showed that the participants were mainly concerned with security issues, but they thought that using NFC is 'cool', comfortable and better than traditional methods. However, the developed solution only aims to solve particular and specific problems.

Furthermore, given that this type of communication typically never occurs at distances above 4 centimeters [Goo13c], the user is able to control and know when a communication is being held, reducing the probability of misleading transactions. This characteristic helps unexperienced users feel safe, and with the power to interrupt an operation whenever they want to.

Regarding security, several authors showed the vulnerabilities of NFC and techniques to exploit the technology in mobile devices [HB, Ker11, VK11, MLK+09]. Hence, if it is necessary to

communicate private information, it is important to define an encryption protocol that secures the transmissions through NFC. These vulnerabilities introduce new and challenging requirements in the process of sharing configurations.

In the scope of this dissertation, a context-based configuration process is intrinsically related with the user's location, conditioning and characterizing the type of configurations that are exchanged. Since a configuration service must provide a physical NFC-enabled system that interacts with the user's device, the environment, the user itself, a social situation or the surrounding infrastructures will define the configurations that are provided in a specific location.

Furthermore, these configurations can also be temporary, i.e., they will only be active while the user is in a specific place, situation or within a defined amount of time. Therefore, we define a temporary and context-based configuration process as a scenario where it is possible to provide configurations that only work when a user is in a well-defined location and temporal frame.

There is an NFC data exchange format for sending data between NFC-enabled devices, but the payload is always defined by the developer [For08]. Additionally, there are protocols for remote configuration of devices, like the one specified by the Open Mobile Alliance for management of mobile devices [All06]. However, the configurations are transported over HTTP. So, there is no data format to send configurations through NFC. Furthermore, these device management protocols are mainly used to massively configure mobile devices, and the information between the server and the mobile device must be synchronized. Thus, these protocols are not suitable for temporary and context-based configurations.

To address these issues, the objectives of this dissertation are defined as follows:

- A generic framework for sharing configurations through NFC between a mobile device and a generic external service;
- Usage of an encryption scheme to guarantee security and authenticity when sending information through NFC.;
- A data format for sharing configurations through NFC;
- A prototype based on the defined architecture;
- Proof-of-concept applications developed with the proposed prototype.

The prototype allows developers to use the service by creating different types of applications and configuration scenarios. The configuration process is handled by the service and all configurations are broadcasted to the right third party applications. Thus, an application can subscribe its intent and receive the respective configuration when a configuration process takes place. These configurations must be active and parameterized, so they can be delivered with success. All information is sent using a specified data format and using an encryption protocol, avoiding security threats to the system.

1.2 Real World Scenarios

In order to further complement the idea that a complex configuration process can be unattractive to users, this Section presents some real world scenarios where our solution could help suppress these problems.

1.2.1 Airport

An airport could provide an easy way to configure Wi-Fi connections through NFC. In a normal scenario, the user would need to manually configure the Internet connection. For doing that, it would be required to know what was the network connection Service Set IDentifier (SSID), the security protocol, the Extensible Authentication Protocol (EAP) method, etc. Moreover, he could be tricked into initiating a connection with a rogue Access Point (AP) and to provide private information - known as evil twin attack [Bib05]. With this solution, and considering that he had an airport configuration application installed, he would only need to bring together his mobile phone and an NFC platform to establish the connection. This method would be faster and it would not require advanced knowledge of the network, because all information would be transmitted safely and used to automatically configure the connection.

1.2.2 Tourism center

Another example would be an application for a tourism center, where a worker could help someone looking for a street or monument by simply sharing the coordinates of the destiny. Normally, the worker would need to search on his computer for the desired place and manually insert the coordinates in the tourist's mobile phone. The worker would need to know how to handle different applications and even mobile phones. With the solution proposed in this dissertation, he would only need to select the coordinates to send on his computer. Then it would be possible to use the NFC interface to send the respective coordinates, and any other information, allowing to configure the device and help the tourist find the desired place.

1.2.3 Train station

Finally, a train station could provide a free application for mobile phones with the latest news of the day. In order to update the news, the user's application would need to have some kind of connection with the news server, provided by the station. There are some communication technologies that could be used, like Bluetooth or Wi-Fi. However, NFC provides an easier and faster way of connecting two devices - this is further discussed in Chapter 2. Therefore, the user would only need to come close to an NFC platform, provided by the station, and have the latest news on his mobile phone without the need for an active internet connection. This scenario does not involve a specific configuration handover, but the application could also represent a user's ticket. Thus, it could take advantage of the authentication process to validate the ticket, while receiving the news.

There are many other applications that could be described as use case. The generic nature of the solution allows to have a wide range of scenarios that provide the final user with solutions that overcome the setup difficulty and that are faster than traditional methods. Moreover, it is possible to develop secure solutions, thus improving the overall experience of the final user.

1.3 Report structure

The remainder of this report is divided as follows: Chapter 2 presents background information about wireless communication technologies, comparing them. Chapter 3 describes and discusses related work done in the context of the proposed solution. Chapter 4 describes the requirements of the solution. Chapter 5 details a solution architecture for the problem identified in this dissertation, including its components and its functionalities. Chapter 6 exposes the developed prototypes for each module, which follow the functional and non-functional requirements defined for each one. Finally, Chapter 7 closes this document with remarks about the developed solution and future work.

Chapter 2

Wireless Technologies

2.1 Near Field Communication

Near Field Communication is a short range, high frequency wireless communication interface that enables data transfer between two devices with an NFC chip. It is considered a subset of RFID with some improvements in terms of security, offering a simple and intuitive way of communication. [Kum10, EA12]

In 2004, Nokia, Philips and Sony created the NFC Forum [EA12]. The main objective was to promote the use of NFC and to develop specifications, ensuring interoperability among devices and services. The forum has now more than 170 members, including manufacturers, application developers and financial services institutions and released 16 specifications to date [For13].

2.1.1 Specification

The main characteristic of NFC is that it is a wireless communication technology that works up to a distance of about 20 cm. The specification for this technology, described by NFCIP-1 (Near Field Communication Interface and Protocol 1), can be found in ISO 18092, ECMA 340 as well as in ETSI TS 102 190. It specifies modulation schemes, codings, transfer speeds and frame format of the Radio Frequency (RF) interface, initialization schemes and conditions required for data collision control during initialization. It declares that all NFC devices must operate at the center frequency of 13.46 MHz [Kum10, EA12, Fis09, Ele06, Int08].

NFC devices can be classified as either active or passive, depending if the device generates its own RF field or uses the power generated by another device. When two devices are brought together, either touching themselves or not, two different configurations are possible, as described in Table 2.1.

The communication between two devices can occur at different data rates, depending on the coding scheme used. The supported data rates are 106, 212 and 424 kBaud. If an active device is transferring at a rate of 106kBaud, then it is using a modified Miller coding with 100% modulation. If the baud rate is greater than 106k, the Manchester coding scheme with a modulation of 10%

Table 2.1: Communication configurations of NFC [Int08].

Communication Mode	Description
Active	The communication takes place between two active devices. Each device generate its own RF to send data.
Passive	There is an active and a passive device. The passive device has no power source and uses the RF field generated by the active device to send data.

is used. A passive device always sends the data using a Manchester coding with a modulation of 10% [Int08].

There are two roles that each device can assume in a communication. Considering that the initiator is the one who wants to communicate with the target, the target first receives a message from the initiator asking for some data. Then, the target sends its reply to the initiator. It is not possible for the target to send any data to the initiator without first receiving a request. A passive device always acts as an NFC target, where an active device who wants to communicate needs to generate the radio field first. It is not possible for two passive devices to communicate with each other [Int08]. Table 2.2 resumes all possible combinations.

Table 2.2: Possible role combinations between Active/Passive and Initiator/Target.

	Initiator	Target
Active	Possible	Possible
Passive	Not Possible	Possible

The NFC Forum also specified a new standard: NFCIP-2 (ISO 21481 or ECMA 352). It is intended to be used by mobile devices that support communication according to ISO 18092, ISO 14443, but are also compatible with other standards like ISO 15693. Therefore, this protocol allows to combine functionalities between NFC and RFID readers. In order to accomplish this, the forum established three different operation modes: Peer-to-Peer Mode, Reader/Writer Mode and Card Emulation Mode [Int10]:

- **Peer-to-Peer**
  In this mode, two NFC devices can exchange data. This operation mode follows the NFCIP-1 standard already described, where there are two communication modes - active and passive.

- **Reader/Writer**
  In Reader/Writer mode, an NFC device communicates with an NFC Forum Tag. These tags don't have a power source, acting always in passive mode. The reader/writer mode on the RF interface is compliant with the ISO 14443 and FeliCa schemes.

- **Card Emulation**
  The NFC device mimics a traditional contactless smart card and can be used with an external

RFID reader. With this mode it is possible to use NFC devices as payment and ticketing methods, without changing the existing infrastructure.

Table 2.3 shows all possible communications between and NFC Forum Device and either an NFC Forum Device, NFC Forum Tag or a RFID Reader/Writer terminal.

Table 2.3: NFC Forum Device communication links [Gal11].

		Communication link between NFC Forum Device in:		
		Peer-to-peer mode	Reader/Writer mode	Card emulation mode
and an NFC Forum Device in:	NFC Forum Peer-to-Peer Mode:	yes	-	-
	NFC Forum Reader/Writer Mode:	-	-	yes
	NFC Forum card emulation mode:	-	yes	-
and an NFC Forum Tag in:	operating as ISO18092 Target:	-	yes	-
	operating as one of the NFC Forum Type Tag Platforms:	-	yes	-
and a Reader/Writer terminal:	-	-	-	yes

2.1.2 Security

Security is defined has the prevention of unauthorized access and unauthorized manipulation of information. It is possible to characterize it in three main categories [Ker11]:

- **Secrecy**: the measures to prevent data from unauthorized access;
- **Integrity**: the measures to prevent data from unauthorized changes;
- **Availability**: the percentage of time for which the data is accessible.

Near Field Communication is a wireless communication interface, so the focus is on the risks of the *Air Interface* of NFC. In all communication modes the following attacks can occur with or without the owner's knowledge [HB, Ker11]:

- **Eavesdropping**
  Near Field Communication communicates over the air, so eavesdropping is theoretical possible. In active mode, eavesdropping can be done up to a distance of 10m, whereas if the sending device is in passive mode, the distance is reduced to about 1m. Kortvedt [KMl10] showed that it is possible to build a low cost eavesdrop equipment.

- **Data Corruption**
  If the attacker has a good understanding of the used modulation scheme and coding, he can send data in the valid frequencies and corrupt the data being transmitted. However, he cannot manipulate the actual data. It is like a Denial of Service attack. He would only compromise the availability of the service.

- **Data Modification**
  In this type of attack, a valid, but manipulated, message is given to the receiver. The attacker would need to know what was the coding scheme for the modulation and when to change it, because the data cannot be changed arbitrarily.

- **Data Insertion**
  In a data insertion attack, the attacker inserts valid messages into the data exchange between two devices. It is only possible if the answering device takes too much time to answer, leaving an open window where the attacker can send some manipulated data to the receiver. If the attacker answers at the same time of the answering device the data stream will overlap, corrupting the data.

- **Man-in-the-middle**
  In a man-in-the-middle attack, two parties are tricked into a three party conversations by a third party, the attacker. Instead of talking with which other, they only talk to the attacker. On the other hand, the attacker receives all the data and decides what to do. Lets assume that two parties, Device 1 and Device 2, connecting to each other are tricked into a three party conversion by an Attacker that detects a communication intent. This scenario is shown in Figure 2.1. The attacker needs to disturb the RF field in order to block the signal from getting to Device 2. If Device 1 is listening to signal disturbances, it will detect this manipulation, ending the conversation. Considering it is not checking for active disturbance, the Attacker needs to send data to Device 2. The problem is the RF field from Device 1 is still there, and when the Attacker decides to generate a second RF field, both fields will become active at the same time. Knowing that it is practically impossible to perfectly align the two RF fields, this setup is impossible.

  If they are using active mode, when device 1 sends the first message, it will turn off his RF field. The problem is that when the attacker sends the message to device 2, device 1 will also receive that message, because it is waiting for the reply from device 2. So, device 1 will detect some problem, stopping the protocol. These kind of attacks are practically infeasible.

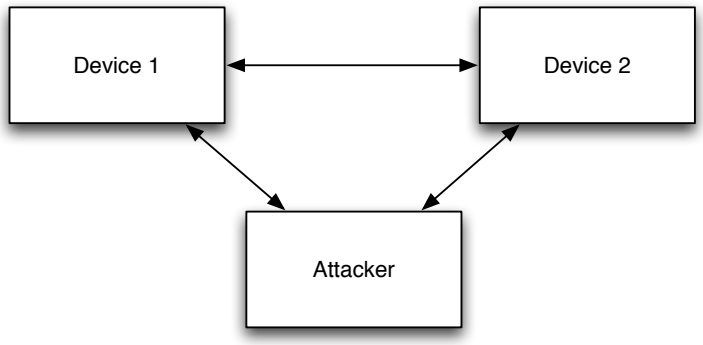

Figure 2.1: Man-in-the-middle setup scenario.

Most of the attacks described can be prevent using a secure channel to communicate. Using an authentication and encryption method, for example, eavesdropping would still be possible, but the attacker could not decrypt the read information. Due to the inherent protection of NFC against Man-in-the-middle-attacks it is possible to setup a secure channel with confidence [HB].

Furthermore, normally it is not possible to read information from an NFC device without asking for permission. As such, an attacker could not steal information while walking by someone with an NFC device in his bag, for instance.

Breitfuß and Haselsteiner [HB] suggest that a key agreement protocol like Diffie-Hellman based on RSA or Elliptic Curves can be applied to establish a secret between two devices. This secret can be later used to derive a symmetric key like 3DES or AES. This key can be used to setup a secure channel providing confidentiality, integrity, and authenticity of the transmitted data.

2.1.3 NFC Data Exchange Format

NFC Data Exchange Format (NDEF) specification defines a binary message encapsulation format to exchange information between an NFC Forum Device and another NFC Forum Device or an NFC Forum Tag. This message can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single message construct [For06]. Type identifiers may be URIs, MIME media types, or NFC-specific types. By itself NDEF does not provide a secure transport of the information. It only encapsulates the application payload, so the application is in charge of the security mechanism, if needed.

An NDEF message is composed of one or more NDEF records. Depending on the application or tag type, a variable number of records can be encapsulated in an NDEF message. In Figure 2.2 shows a general view of an NDEF message.

2.2 Other wireless technologies

There are other wireless technologies that can provide a connection between two devices and that could be used to develop a solution for a remote configuration service of mobile phones. Thus,

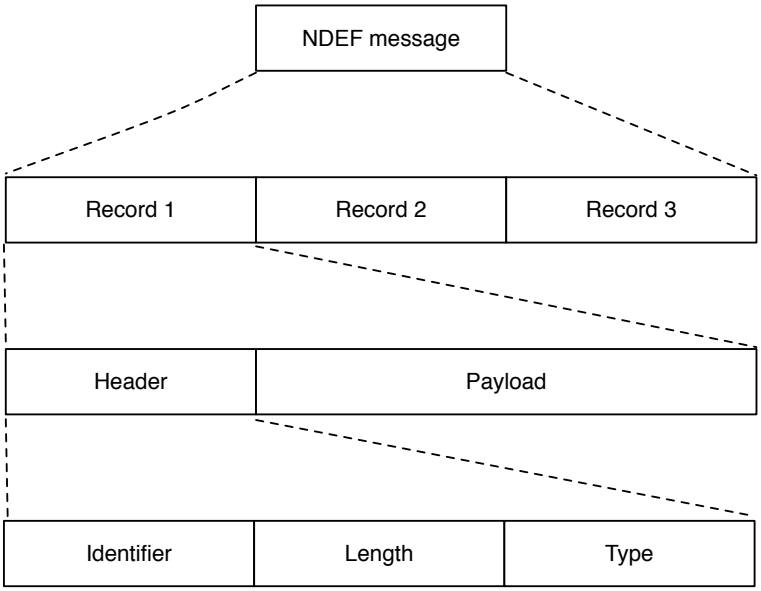

Figure 2.2: General view of an NDEF message [For06].

this Section describes related technologies and compares them with NFC. The main objective is to compare different wireless technologies and to choose the most suitable one to establish a easy, fast and intuitive communication link between two devices.

2.2.1 Wi-Fi Direct

Wi-Fi, short for Wireless Fidelity, is a wireless technology that allows the exchanging of data wirelessly at high-speed. It is mainly used in devices that exchange data over a computer network, like the Internet. Any Wi-Fi-enabled device can connect to the Internet via an Access Point (AP). The Wi-Fi Alliance is responsible for promoting this technology and to certificate products if they conform to certain standards of interoperability [Web13].

Recently, Wi-Fi Direct introduced a new concept where Wi-Fi devices can connect to each other without the need of a wireless access point. This standard permits direct transference of data between two devices with minimal setup - it uses the Wifi Protected Setup (WPS) to establish a connection. In order to setup the connection, normally we only need to bring the devices together and then trigger a pairing process [All13]. It operates at speeds up to 250Mbps, has a range of 100 meters and requires that the user knows the device name he wants to connect to.

2.2.2 Bluetooth

Bluetooth is a wireless technology that operates at short distances and that enables two Bluetooth-enabled devices to securely connect to each other. This technology allows sharing voice, music, photos, videos, and other information wirelessly between two paired devices [SIG13]. There are two ways of pairing devices. In Bluetooth v2.0 and before, each device must enter a PIN code and

if both devices enter the same PIN, the pairing is successful. After Bluetooth v2.1, the user only needs to confirm the pairing process by comparing a 6-digit numeric code on each device. Both mechanisms permit the creation of a secure channel for communication. More recently, Bluetooth Low Energy (BLE), a sub-set of Bluetooth 4.0, was launched with a set goal to reduce power consumption and cost. This new version is characterized by an over the air data rate of 1Mbps, a maximum working distance of 50m and a setup time of about 3ms. BLE is still in adoption by manufacturers, thus not being widely available at the moment.

Bluetooth technology was invented in 1994 by engineers at Ericsson. In 1998, a group of companies agreed to work together using Bluetooth as a way to connect their products, thus forming the Bluetooth Special Interest Group (BSIG). This organization maintains and controls the main technical aspects of the technology [SIG13].

2.2.3 IrDA

IrDa (Infrared Data Association) is a set of protocols for wireless infrared communications that uses point and shoot principles to transfer data between devices. It provides a low bit error rate, a physically secure data transfer, and does not require a pairing process. The devices need to be perfectly aligned and in line-of-sight of each other in order to create a stream for data exchange [Inf97]. No additional setup steps are required. Nowadays, it has fallen into disuse, mainly because of the need of a direct line of sight, being replaced by other wireless technologies such as Wi-Fi and Bluetooth that can support a wider variety of hardware, like mouses and keyboards.

2.2.4 Comparison with NFC

Table 2.4 compares Wi-Fi Direct, Bluetooth (v2.1 and v4.0), IrDA and NFC in terms of network topology, range, speed, setup time and costs.

Table 2.4: Comparison between NFC, Bluetooth, IrDA and Wi-Fi Direct [Ele06, Kum10, Hop09].

	NFC	**Bluetooth**	**IrDA**	**Wi-Fi D.**
Topology	P2P	Point-to-multi-point	P2P	P2P
Range	0.2m	100m (v2.1), 50m (v4.0)	1m	100m
Speed	424Kbps	1Mbps	115Kbps	250Mbps
Setup time	<0.1s	6s (v2.1), 3ms (v4.0)	0.5s	Depends
Setup effort	Low	High	Moderate	Moderate
Cost	Low	Moderate	Low	High

In terms of network topology, all of the technologies could be used to develop a solution for the problem of this dissertation, because all of them allow the connection of at least two devices for sharing data.

Regarding range, all of the technologies can provide a close interaction for sharing information. Furthermore, NFC and IrDA require the user to be close enough and that he is seeing both devices connecting with each other, decreasing the possibility of impersonation attacks.

In the scope of this dissertation, the amount of data being exchanged between the devices is not very large, thus the connection speed is not a priority and all technologies can be used with that principle in mind. Considering a scenario where we use NFC and we define a setup time of 3 seconds, it is possible to send up to 159kB of information. For instance, a regular Wi-Fi Protected Access 2 (WPA2) configuration data has a size of approximately 2.5kB.

The setup time is one of the most important characteristics to consider. NFC and Bluetooth exceed the other technologies with a low setup time. However, one of the most important objectives of this dissertation is to minimize the setup effort during the configuration, and NFC is the only technology that allows triggering the setup and connect two devices by simply bringing them close.

At last, the cost associated with NFC is one of the lowest among all described technologies, corroborating once more the use of it in the scope of this dissertation.

2.3 Summary

This Chapter introduces Near Field Communication, a short-range technology that allows simple and intuitive interactions between two NFC-enabled devices. This communication link has some security threats that can be prevented by using a secure channel. The NFC Data Exchange Format, a binary format, is also standardized and mainly used to send information through NFC.

NFC is one of the most fittest technologies to provide an easy and fast way of connecting two devices, mainly because (i) setup time and (ii) associated cost. It is expected that in the future the use of this technology increases and that, ultimately, it can be available in every mobile device [Hat11].

Chapter 3

Related Work

This Chapter describes different studies and projects regarding remote configuration of devices through NFC. It is divided in four Sections: device management protocols, mobile phone configuration using NFC tags, remote configuration of devices using a mobile phone and pairing of devices using NFC tags. Each Section specifies relevant work to the topic of this dissertation, showing limitations and differences when compared to the objectives of this dissertation, namely if they (i) use NFC, (ii) aim mobile device configuration, (iii) implement a security layer, (iv) define a configuration protocol and (v) provide a service for developers. A full comparison is presented in Table 3.1.

3.1 Remote Device Management Protocols

As the main goal of the solution presented in this dissertation is to provide a service for remote configuration of mobile devices, this Section details two protocols for remote device management, answering the following questions: *Can these protocols be used with NFC? Can they provide context-based configurations for a specific device?*

3.1.1 Open Mobile Alliance Device Management

The Open Mobile Alliance (OMA) Device Management [All08] is a protocol that allows third parties to configure mobile devices on behalf of the end user. An external party, such as a wireless operator, a service provider or a corporate information management department, can remotely set parameters, conduct troubleshooting servicing of terminals, and install or upgrade software.

This protocol uses a sub-set of XML, defined by SyncML, for data exchange. The communication protocol between the server and the device being managed was designed to support different data transports. The communication can be started by both ends, and a sequence of messages

Table 3.1: Related work summary.

	Uses NFC	Aims mobile device configuration	Has a security layer	Defines a configuration protocol	Provides a service for developers
Open Mobile Alliance Device Management		✔	✔	✔	
Broadband Forum - CPE WAN Management			✔	✔	
InstaWifi	✔	✔			
NFC Task Launcher	✔	✔			
AppNearMe	✔	✔			✔
Means for provisioning and managing mobile device configuration over a near-field communication link	✔	✔			
Secure Hotspot Authentication through a Near Field Communication Side-Channel	✔	✔	✔		
Zero-Configuration of Pervasive Healthcare Sensor Networks	✔				
Design and Evaluation of a Telemonitoring Concept Based on NFC-Enabled Mobile Phones and Sensor Devices	✔				

might be exchanged in order to successfully configure the device. A secure mechanism with authentication is built-in to ensure that the server and the device can communicate safely and only after proper validation.

OMA-DM Protocol consists of two parts: setup phase (authentication and device information exchange) and management phase, that can be repeated as many times as the server wishes. The content received from the server can contain specific configurations and determine if the session must be continued or not. The response package from client starts a new protocol iteration. This process can take an unpredictable amount of time to complete, because there is no timeout between packages. Figure 3.1 resumes the two phases.

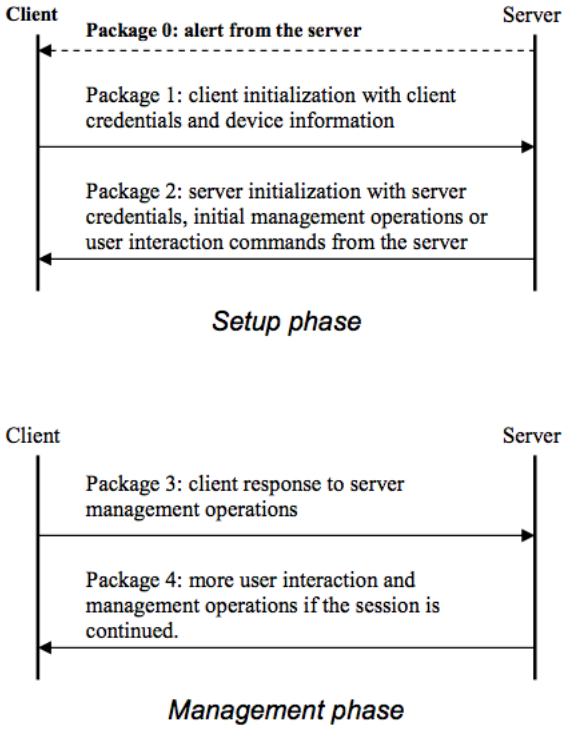
Client
Server
Package 0: alert from the server
Package 1: client initialization with client credentials and device information
Package 2: server initialization with server credentials, initial management operations or user interaction commands from the server
Setup phase
Client
Server
Package 3: client response to server management operations
Package 4: more user interaction and management operations if the session is continued.
Management phase

Figure 3.1: Overview of OMA DM two phases protocol - setup and management phase.

3.1.2 Broadband Forum - CPE WAN Management

The Customer Premises Equipment (CPE) Wide Area Network (WAN) Management [For07] is a protocol that specifies a secure mechanism of auto-configuration between a CPE and an Auto-Configuration Server (ACS). This mechanism includes auto-configuration and dynamic service provisioning, software/firmware image management, status and performance monitoring and diagnostics.

The configurations shared between the server and the device are transported over HTTP - or HTTPS, guaranteeing security - using SOAP (Simple Object Access Protocol). If the CPE is not authenticated using SSL (Secure Sockets Layer)/TLS (Transport Layer Security), the ACS authenticates the CPE using HTTP. If SSL/TLS is being used for encryption, the ACS can use either basic or digest authentication. Finally, if SSL/TLS is not being used, then the ACS MUST use digest authentication. In the example shown in Figure 3.2, the ACS first reads a set of parameter values, and based on the result, sets some parameter values.

3.1.3 Discussion

None of these protocols specifies the use of NFC to share configurations between a server and a device. They both require that the current status of the device is known at the servers, adding extra synchronization needs. They don't fit well in situations where the configuration is temporary or specific to the context and the device. They are usually used to massively configure a great number

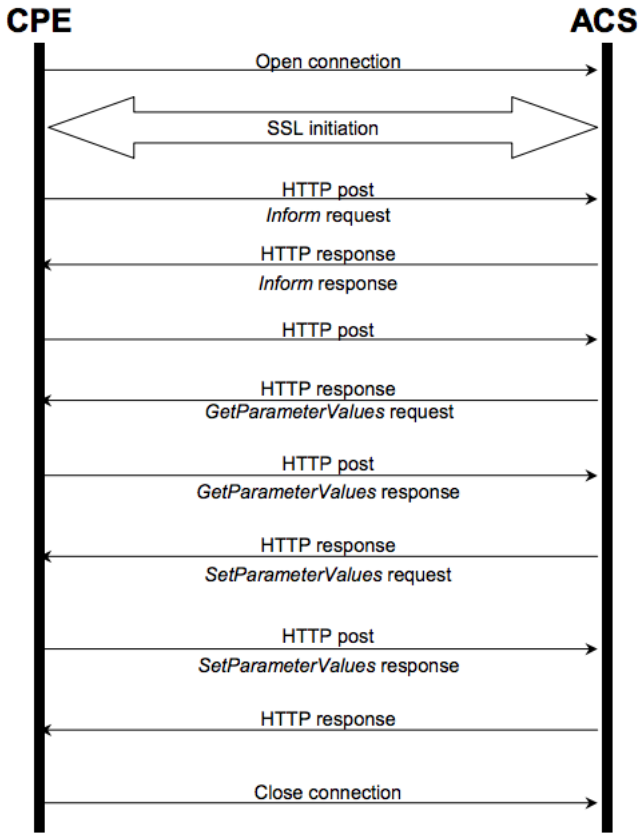

Figure 3.2: Example of a configuration process using the Broadband Forum - CPE WAN Management protocol.

of devices, whereas the configurations that will be sent to the device using the prototype of this dissertation are normally very context-specific.

The CPE WAN Management Protocol, in particular, is only used to configure WAN devices like routers and switches and not mobile phones.

3.2 Mobile Device Configuration using NFC

Nowadays, it is already possible to configure mobile devices through NFC. However, and considering the scope of this dissertation, some questions arise: *Do they allow to configure third party applications? Do they provide an external service where we can manage the configurations? Is it possible to have multiple configurations at the same time? Do they implement a security mechanism for exchanching information through NFC?*

3.2.1 InstaWifi

InstaWifi [Che13], presented in Figure 3.3, is an Android application to share Wi-Fi connection settings through NFC and QR Codes. After installing it, the network owner can configure NFC tags or generate a QR Code with its network settings. In order to configure the network in another

device, it is necessary to also install the application in it. Then there are three ways to perform the configuration: touching both mobile phones, reading the NFC tag, or using the QR Code that was generated in the network owner mobile phone. The first two options use NFC (Android Beam) and the last one uses the camera. InstaWifi supports only two Wi-Fi security protocols: WEP and WPA/WPA2.

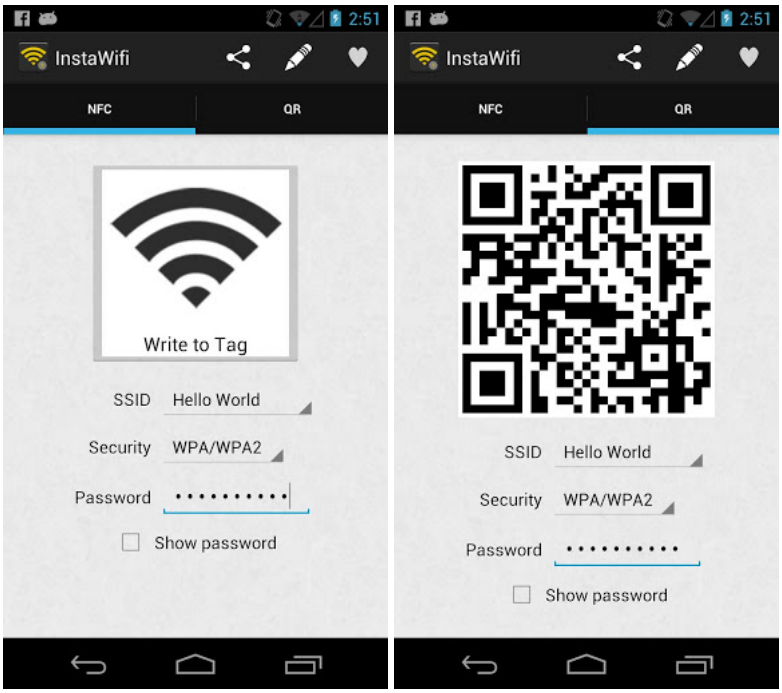

Figure 3.3: InstaWifi screen shots showing the interface for writing a Wi-Fi configuration in an NFC tag or sharing it via an auto-generated QR Code [Che13].

3.2.2 NFC Task Launcher

NFC Task Launcher [Tag13a, Tag13b], presented in Figure 3.4, is an Android application that allows the user to switch system settings (Wi-Fi, Bluetooth, Airplane mode, etc) on and off using NFC Tags. It is possible to choose which settings the tag will trigger. After writing to the tag, the user only needs to touch the mobile phone to it, switching the state of the chosen settings. It also allows to check in in different applications like Google Latitude, Foursquare, Facebook and Google Places, to send tweets written in the NFC tag, to start and stop media files, to automatically tether a laptop to a mobile phone and to share Wi-Fi passwords.

3.2.3 Means for provisioning and managing mobile device configuration over a near-field communication link

This patent [Abe06], created by Miller T. Abel, specifies an NFC-enabled device provisioning and configuration system, capable of pushing device configurations and service indicators through NFC. The device is capable of exposing its management interfaces across the NFC channel and can

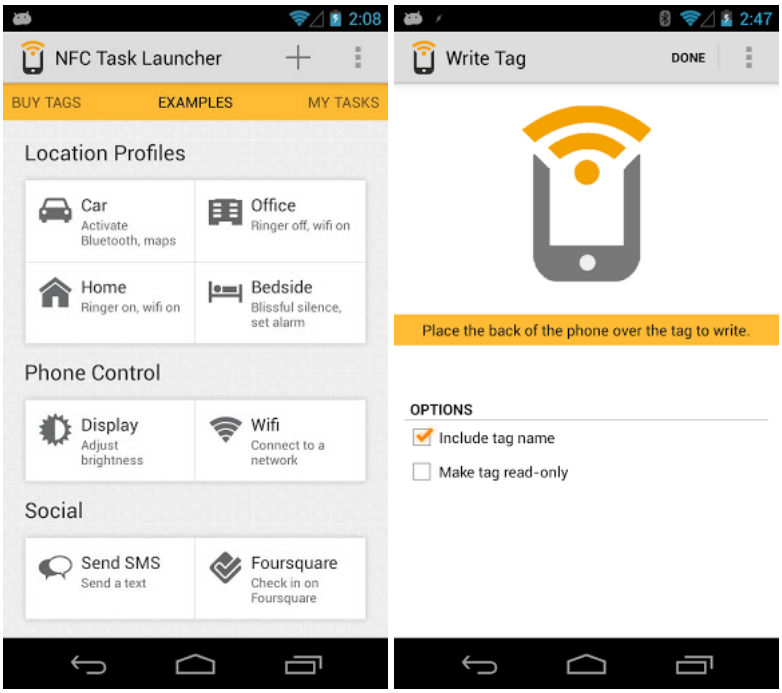

Figure 3.4: NFC Task Launcher activities for choosing a pre-defined task and for writing the selected set of configurations in an NFC tag [Tag13b].

be in factory-configured state without general connectivity. The patent claims 'a device comprising radio frequency circuitry that facilitates near field communication' [Abe06] and 'a server that establishes a near field communication channel with the device' and that 'provides provisioning information to the device via the near field communication link' [Abe06].

3.2.4 Secure Hotspot Authentication through a Near Field Communication Side-Channel

Matos, Romao and Trezentos specified an architecture for a Wi-Fi hotspot authentication system based on an NFC side-channel to address the security issues of these configurations and access in public locations [MRT12]. The system relies on embedding network and security information in NFC enabled devices that store access point information and allow the user to connect to secure wireless networks. This approach prevents several security threats such as an evil twin attack, hotspot or captive portal eavesdropping, and Man-in-the-Middle attacks.

The developed solution supports both static configurations - using NFC tags - and NFC peering - uses the NFC channel to negotiate access keys, possibly using two-way authentication. The results show that the prototype allows an above average bandwidth channel and supports very short association times - under 49ms.

3.2.5 Discussion

InstaWifi and NFC Task Launcher allow the automation of tasks using NFC tags and to share Wi-fi connections between mobile phones running Android. Moreover, the proposed architecture by Matos, Romao and Trezentos prevents several security threats and offers a fast way of configuring Wi-Fi connections through NFC. However, none of the projects answer to all objectives of this dissertation. They do not offer a service for programmers, cannot be controlled by an external service, and do not allow the configuration of third party applications. In particular, InstaWifi and NFC Task Launcher do not encrypt the saved information on the tags. Nevertheless, their approach to configuration of mobile phones using NFC shows that it is possible to use the technology to design a generic and parameterizable service for remote configuration of mobile phones.

Regarding the patent created by Miller T. Abel, it specifies the basic notions of a generic system and its devices for remote configuration through NFC, thus not referring a detailed architecture for the system. The communication between the server and the device is only unidirectional and does not have an authentication system. It only provides a way of provisioning information, and not a generic service for external configuration of mobile phones. Moreover, it does not define a security protocol, a template for the configurations and assumes that the device already has a built-in compatibility with the service.

3.3 Remote Configuration of Devices using a Mobile Phone

This Section presents a framework that allows the user to configure a device via an external service using NFC. Although not providing a service to remotely configure a mobile device, it presents a similar idea, and, therefore, it is analyzed here.

3.3.1 AppNearMe

The main objective of AppNearMe [App13], shown in Figure 3.5, is to provide a simple way to configure devices that don't have an user interface using a mobile phone running Android. With this framework the user can create an application using HTML5/Javascript that reads the information sent by the device, waits for him to configure the settings and finally pushes the information back to the device. The framework only works with specific NFC readers.

3.3.2 Discussion

AppNearMe provides an external service to configure devices using a mobile phone through NFC. However, this dissertation is trying to find the best architecture to accomplish the opposite, i.e., an external service for remote configuration of mobile devices. Moreover, this framework does not encrypt the information shared between the devices and does not define a protocol to share configurations.

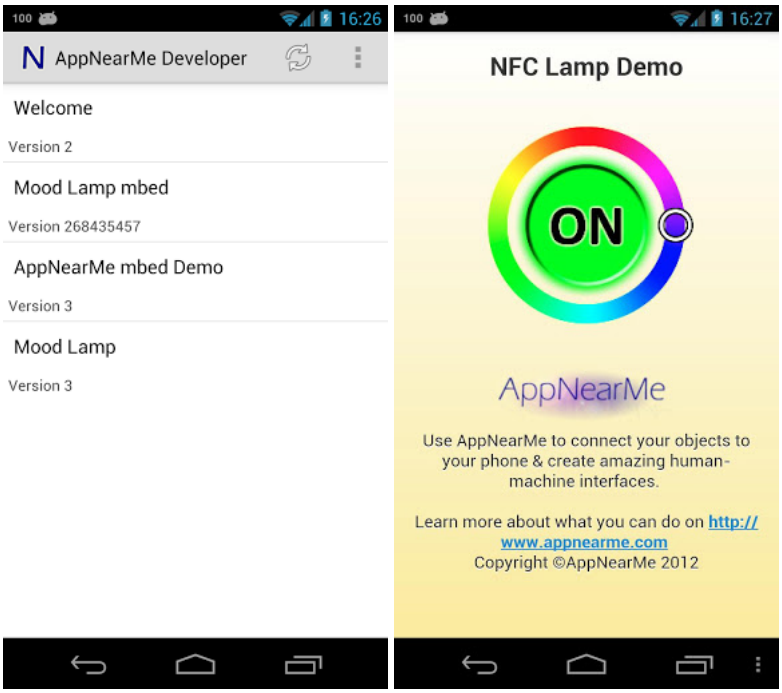

Figure 3.5: AppNearMe Developer activities for managing the current device configurations and for controlling an external lamp state and color through NFC. [Tag13b].

3.4 Pairing of Devices using NFC tags

The works presented in this Section have a common goal: provide a mechanism to pair devices through NFC. Although their main objective is not to offer a remote service for configuration of mobile devices, it is still possible to analyze their main conclusions and if they implement any kind of security protocol for the communication link.

3.4.1 Zero-Configuration of Pervasive Healthcare Sensor Networks

In 2008, a research project at the Engineering College of Aarhus was trying to achieve a zero-configuration system for a health care sensor network [Wag08]. The prototype developed is able to read NFC-tags of sensors and patients, coupling both using a SOAP based web service over the network (depending on device support). Since patients can be placed in close proximity, the pairing process guarantees that there is not another station picking up the wrong signal, like the previous solution that used Bluetooth technology. The project was developed using the Nokia Software Development Kit for Nokia 6131, one of the first mobile phones having NFC. To use the application, the caretaker has to make the mobile phone touch the patient NFC-card and then do the same with the sensor NFC-tag. The system automatically sends the data to the server, linking the sensor to the patient profile.

3.4.2 Design and Evaluation of a Telemonitoring Concept Based on NFC-Enabled Mobile Phones and Sensor Devices

Morak J. et al [MKH+12] developed a solution to configure a connection and to share data between a mobile device and wireless sensor devices using NFC and Bluetooth. The NFC-based system is only used to establish the connection, because it was not optimal to oblige the user to steadily keep the mobile phone close to the NFC-enabled sensor device as long as data were being transferred. Therefore, the user only needs to bring the mobile phone and the sensor close for a few seconds without manually configuring the communication link. This communication link is provided by Bluetooth, in order to allow bigger data transfers. The performed tests showed high usability and effectiveness in different scenarios.

3.4.3 Discussion

As described, both projects use NFC to establish a connection between two different objects. They show that NFC-based systems are capable of providing simple, yet effective and fast ways to configure devices, although none of the projects tried to create a generic and parameterizable service to configure mobile phones remotely and they don't implement any kind of authentication method.

3.5 Conclusions

It is easy to observe that none of the above related work covers the goals set in this dissertation, namely (i) a generic service for remote configuration of mobile devices and (ii) a specific format for sharing configurations through NFC. AppNearMe is the only that provides a framework that: (i) brings the control to the user and (ii) lets him extend the system, although it tries to solve a different problem.

The device management protocols do not provide an answer to our goals, namely in terms of (i) context-based configurations, (ii) extensibility and (iii) a parameterizable external service.

Applications that allow the configuration of a mobile phone using NFC tags are starting to emerge, but are limited to the built-in functionalities and can not be extended by the developers or the users.

Some studies [MKH+12, Wag08] suggest that an NFC solution for the pairing of devices results in: (i) high usability and (ii) effectiveness. Although the objectives defined for the dissertation are different, these studies show that it is possible to reduce the user intervention through NFC.

Chapter 4

System Requirements Specification

This Chapter describes the requirements used in order to evaluate if the purpose of this dissertation is successfully achieved. Use cases have become a common practice for capturing functional requirements. Thus, Section 4.1 enumerates a set of use cases, whereas Section 4.2 defines non-functional requirements that a solution shall take into account.

4.1 Use Case Model

The use case model helps to graphically define all interactions between an actor and a system, in order to accomplish an objective. This Section describes all actors, their goals and interactions between use cases.

4.1.1 Actors

An Actor here represents a class of users, roles users can play, or other systems. A complete set of use cases defines all the different ways to use the system, and therefore all behavior required of it, bounding its scope.

In particular, this system shall have four main actors:

- **User**: represents the owner of an NFC-enabled device being configured;
- **Device**: represents a user's device being configured;
- **Third Party Application Owner**: represents a third party application that uses the service to automatically configure his application and that is installed in the device;
- **Service Administrator**: represents someone who is responsible for managing the configuration service. A Service Administrator can also be a Third Party Application Owner.

These actors interact with two main systems: the NFC Service Provider and the Device. The first one shall be responsible for managing and making available all configurations. The second one, shall represent a device being configured.

4.1.2 Service Provider

The Service Provider shall offer an interface where a Service Administrator can manage the service. Furthermore, a Device shall be able to query the service for configurations. The following use cases shall be supported at a Service Provider and are depicted in Figure 4.1:

UC001 Manage Configurations
A Service Administrator should be able to add, edit or remove configurations from the system.

UC002 Manage Values
A Service Administrator should be able to add, edit or remove all values of a configuration. This value represents a configuration parameter that shall be sent to the Device when there is a new request.

UC003 Manage Users
A Service Administrator should be able to add, edit or remove users from the system.

UC004 Set Associated Users
A Service Administrator should be able to select of all users, the ones that have access to a configuration.

UC005 Manage Active Configurations
A Service Administrator should be able to add, edit or remove users from the system.

UC006 Get Configurations
A Device should be able to fetch configurations from the service by providing a valid authentication data. The service provider shall process it and only select the configurations that can be sent to that request.

4.1.3 Device

Both User and Third Party Application Owner interact with the Device. User interaction shall be as minimal as possible, considering that one of the main objectives is to achieve user minimal interaction. However, a User shall feel safe and in control of the process. On the other hand, a Third Party Application Owner shall also be able to use the service without the need to specify a communication layer or how a Device sends a configuration request to a Service Provider and analyzes it. This operations shall be transparent. Therefore, the following set of use cases for the Device are presented in Figure 4.2 and shall be taken into account when idealizing a solution:

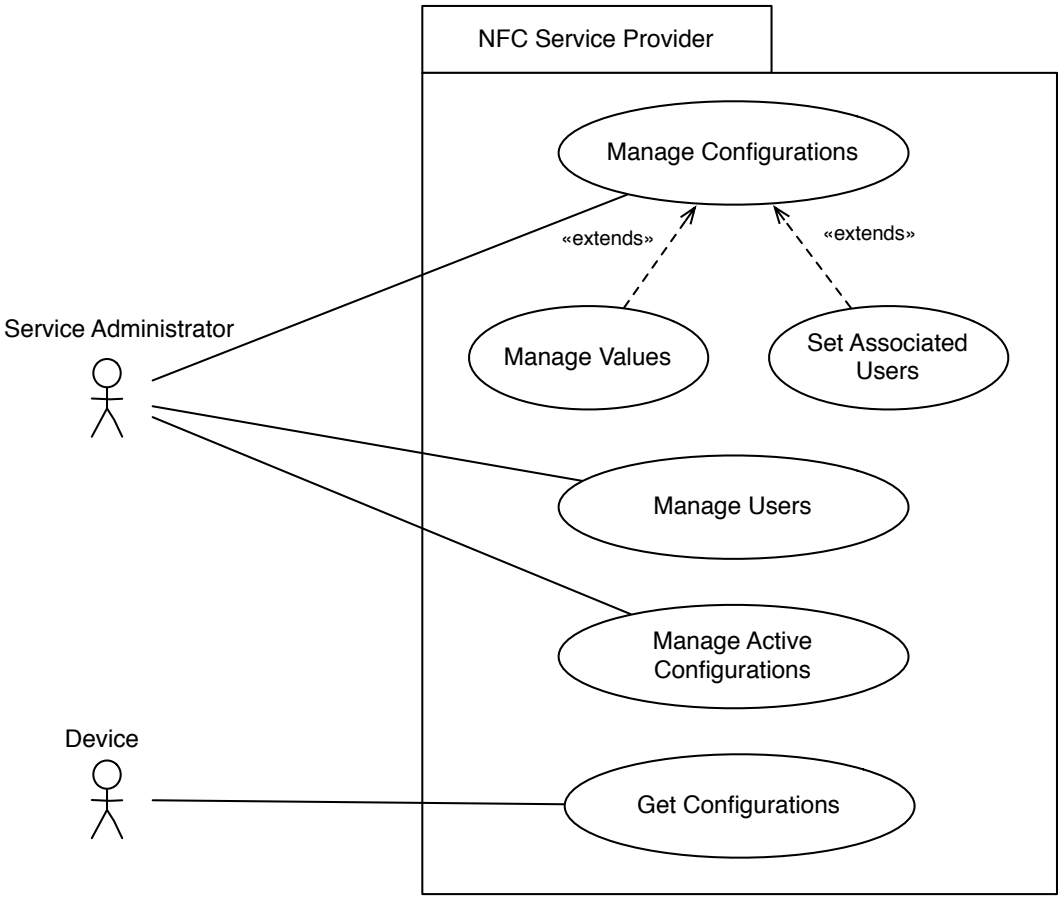

Figure 4.1: NFC Service Provider's use case diagram.

UC007 Manage Account

A User should be able to add, edit or remove an account from the Device.

UC008 Configure Device

A User should be able to configure a device by approaching it to an NFC Service Provider and authorizing the configuration process.

UC009 Subscribe Configuration

A Third Party Application Owner should be able to subscribe an intent of receiving configurations for a specific third party application. The Device is then responsible for providing all configurations to the right applications.

Figure presents a diagram with these use cases and their interaction with a Device.

4.2 Non-Functional Requirements

This Section enumerates a set of non-functional requirements that the solution presented here shall guarantee. Also known as qualities of the system, they evaluate the overall operation of the system.

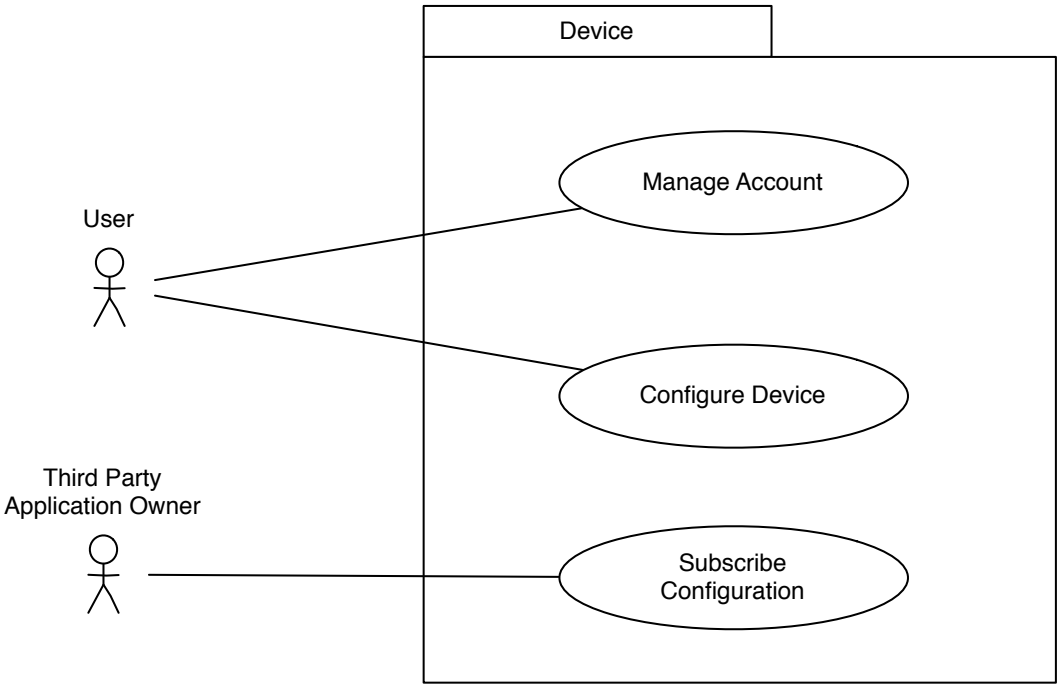

Figure 4.2: Device's use case diagram.

4.2.1 Extensibility

The system shall provide the right tools to add new communication links between a Client and a Service Provider as painlessly as possible. This property ensures a system that tends "*...to have greater longevity, avoiding the expensive cycle of procuring large, fixed systems and then throwing them away when the world changes*"[1].

4.2.2 Usability

Usability shall be taken into account from the beginning. As stated before, a minimal user intervention process is desirable. Therefore, the system experience should be user-friendly, with minimal interfaces and without unnecessary options and screens.

4.2.3 Security

The system shall implement the right mechanisms to secure the information saved and shared between modules. The access to configurations shall be restricted and handled with care. The authentication data should not be exposed. The user shall feel safe configuring is device with the service.

[1] http://www.nfprequirements.org/requirements/non-functional-requirements/altering-systems/extensibility

4.2.4 Timeliness

Since one of the objectives aims to reduce the configuration time, the system should respond on average in less than 3 seconds per configuration. When a user opens the application and brings the device close to a Service Provider, the configuration process shall immediately start and take as little time as possible to respond back with the configurations. After receiving them, the client application has to rapidly broadcast each configuration to the right third party application.

4.3 Summary

This Chapter starts by presenting the functional requirements - as use cases - that a solution shall follow in order to accomplish the proposed objectives. Four main actors are also introduced: (i) User, (ii) Device, (iii) Third Party Application Owner and (iv) Service Administrator. They interact with (i) the Device being configured and (ii) the Service Provider that saves and provides the configurations.

Furthermore, the aforementioned non-functional requirements shall not be ignored when developing a solution. Although not representing concrete functionalities of a system, they permit the description of requirements that will improve the overall quality of a solution.

Chapter 5

Architecture and Design of the Prototype

This Chapter details the architecture developed to address the problems identified in this dissertation, including its components and its functionalities. Figure 5.1 represents a block diagram of the proposed solution. It is composed of three independent systems: Main Server, Local Server and Client. The communication between a Local Server and a Client is performed through a secure Near Field Communication channel. On the other hand, the communication between a Local Server and the Main Server uses the Hypertext Transfer Protocol Secure (HTTPS). All configurations and users are managed by the Main Server. A Local Server is responsible for managing all Client requests, guaranteeing transparency, security and secrecy. Finally, a Client represents the mobile phone being configured. The Client application must be installed in the device, as well as all third party applications that support and want to use the service.

5.1 Main Server

The Main Server provides an online interface and a Representational State Transfer (REST) service that uses JavaScript Object Notation (JSON) as data format. It stores all configurations and users. A configuration entry can have multiple values and associated users. Each user can be associated with one or more configuration. Figure 5.2 represents a class diagram for the Main Server.

A configuration entry is characterized by a name, a description of the application, an application package that is used in the mobile phone application, and three different booleans that (i) define its current state, (ii) if it requires an email information and (iii) if it requires an associated password.

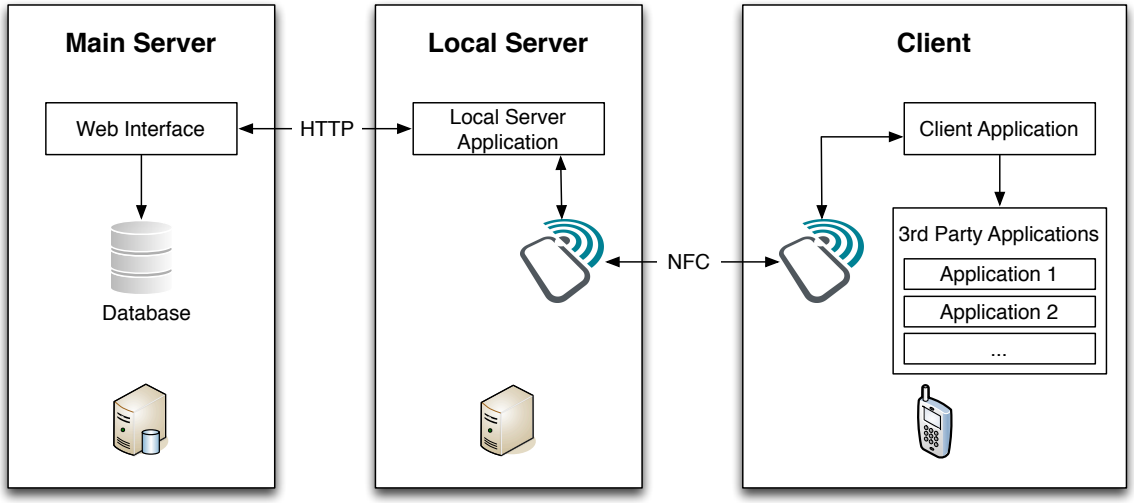

Figure 5.1: Architecture overview diagram. It can be decomposed in three modules: the Main Server, where all configurations are saved and managed; the Local Server, responsible for connecting the Client and the Main Server; the Client, a mobile phone application that provides configurations to third party applications.

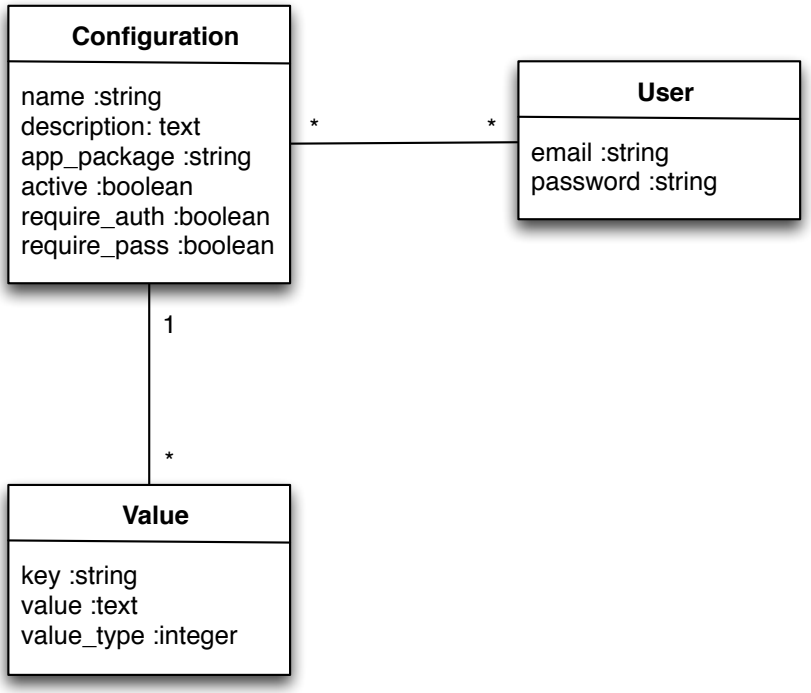

Figure 5.2: Class diagram of the main server. Each configuration can have multiple values and users associated. Each user can be associated to multiple configuration. A user is characterized by an email, which is mandatory and a password, which is optional.

A value, which is associated with a particular configuration, has a key, representing its name; a value, representing the value for that key; and a value type, which is an integer that represents the data type of all values. Section 5.4 details all available types.

In order to insert a new user in the database, an email must be provided. The password field is optional.

At any given time it is be possible to manage the current active configurations. It is possible to manage all configuration values and select the users that can have access to them. Furthermore, the Main Server provides a list of all configurations. It also allows to select which configurations are active and if they require any kind of authentication.

Vendors can decide if their configuration should require any kind of authentication, or should allow unauthenticated access instead. If a vendor decides that a specific configuration does not require authentication, the configuration will be available to everyone, even if there are users associated to it. If an authentication method is defined, only the associated users will receive that specific configuration. Finally, the configuration may require a password, or not. This characteristic allows configurations that require only a username, require both username and password, or require no authentication.

When the server receives a get configurations request, and based on the provided authentication data, a response with the available configurations for that user is sent. This mechanism guarantees that of all active configurations, a user only receives the ones he has access to, following the rules previously presented.

Therefore, if a developer wants to configure his application, he needs to add the application main information, its configuration values and choose the users that can access that configuration. He can also allow access to a specific group of users - that will be used to filter the access if an authentication method is defined.

5.2 Local Server

The Local Server can be described as a bridge between a device being configured and the Main Server. It communicates with the Main Server via HTTPS (Hypertext Transfer Protocol). On the other hand, the communication with all the clients is established via a secured NFC channel.

This middle layer between the Client and the Main Server allows scenarios with more than one Local Server. Each Local Server can communicate with the same Main Server, providing the same information concurrently. Furthermore, this architecture allows to add other Local Servers that would communicate with the Client application through another wireless interface. Considering this, the presence of a Local Server guarantees an expandable and multifaceted architecture.

The configuration process takes place in three steps. First, the Local Server receives an intent from a client to configure his device. Secondly, and after decrypting and checking that the intent came from a valid client, the Local Server communicates with the Main Server. At last, if there is any valid configuration for that client, the information is encrypted and sent back to the client through a secure NFC channel. The information is then handled by the Client application. Figure 5.3 resumes this process.

Although Chapter 4 does not directly specifies this module, this approach meets the defined use cases. Instead of having an NFC Service Provider, this solution has two modules that represent

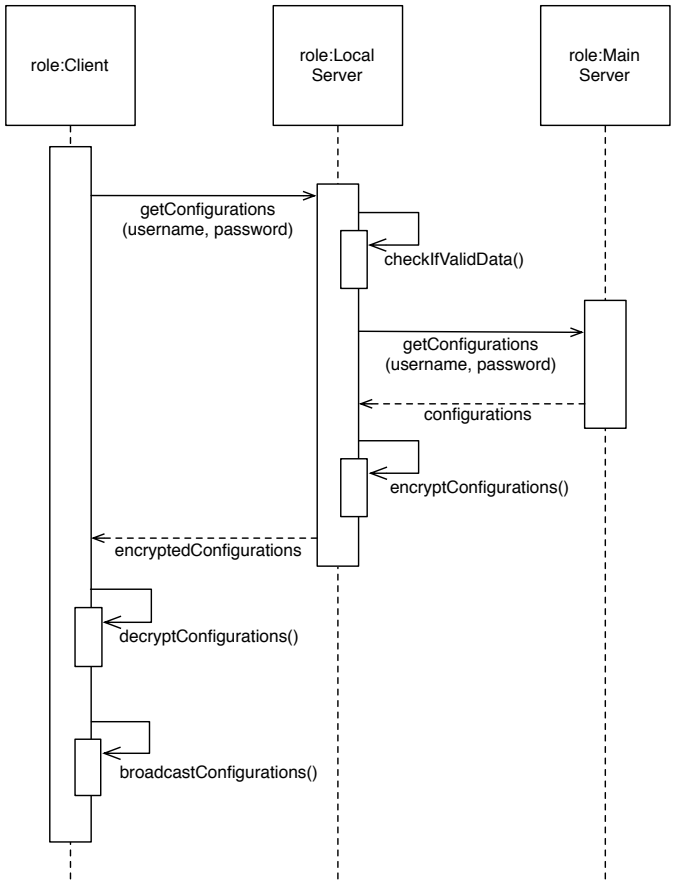

Figure 5.3: Sequence diagram of a generic configuration process.

it. In practice, this decomposition is transparent to the final user and to the third party application owner, thus not changing the way both ends interact with the system.

5.3 Client

The Client represents the device being configured. It interacts physically with a Local Server, requiring minimal user intervention. In order to use the service in a specific mobile device, the client application must be provided natively or installed on the device by the user. Moreover, the Client application broadcasts each received configuration to the respective third party application, so it is also necessary to install all the applications the user wants to configure. After that, the user can provide his authentication, that will be used to fetch the configurations from the Main Server, via the Local Server.

To configure his device, the user must open the Client application and bring the device close to the Local Server NFC reader. The process will automatically start and if there is any configuration information compatible with his authentication data, all third party applications will be informed and configured.

After receiving the respective configuration, a third party application should know what parameters to expect and how to handle them internally. The framework is only responsible for receiving, decrypting and providing the right configurations to the right applications, without reading or saving the data passed.

5.3.1 Subscribing a Configuration

As described in Section 5.1, a configuration has a field that contains its application package. Therefore, only the third party mobile application that defined it as its application package will be informed and have access to that configuration. Moreover, the interested application should subscribe an intent for being informed when there is a new configuration being shared by the Client application.

All configuration values are bundled together and made available to the third party application, that should know the expected values beforehand. Thus, the application can obtain and use them as wished. The responsibility of this process is entirely of the receiver application.

5.4 Data Exchange Protocol

JSON (JavaScript Object Notation) is the data format chosen for sharing configurations through NFC. It is a lightweight format commonly used for data interchanging, it allows to create collections of name/value pairs and ordered list of values. Using JSON also allows to easily extend to other communication technologies. Moreover, the typical response is composed of a JSON Array with multiple JSON Objects, where each one represents a particular configuration and can contain multiple name/values pairs.

A configuration value can assume one of the following types, viz. (i) String, (ii) Integer, (iii) Float, (iv) Array of Strings, (v) Array of integers, (vi) Array of floats and (vii) URL, that fetches the URL content each time the configuration is requested in the Main Server.

In a lower layer, the NFC Forum specifies a data format for sharing information through NFC, the NFC Data Exchange Format (NDEF). It is a lightweight binary message format designed to encapsulate one or more application-defined payloads into a single message construct. An NDEF message contains one or more NDEF records, each carrying a payload of arbitrary type and up to $2^{32} - 1$ octets in size. Since the Local Server interacts solely with the Client application, each message only has one NDEF record. The expected size for a configuration data is of approximately 3kB, representing an NFC transmission time of 6ms.

5.5 Encryption Protocol

In order to guarantee that the information is not tampered and that each party is reliable, a public-key encryption mechanism [RSA12] should be implemented - also known as asymmetric cryptography. In an asymmetric crypto-system, the public key is associated with a private key. The public

key may be known by anyone and it can be used to verify a digital signature that is signed by the corresponding private key and encrypt data that can only be decrypted using the corresponding private key.

The Local Server and the Client applications each have a pair of cryptographic keys (K_{LS}, K_{LS}^{-1}) and (K_C, K_C^{-1}) – a public encryption key and a private decryption key. The private-keys are distributed natively in the code of each application. The public-keys are publicly available in each module.

The Client also generates a symmetric key using the Advanced Encryption Standard (AES) [Nat01] - a well-respected symmetric algorithm. This algorithm provides a symmetric block cipher that can encrypt and decrypt information. The encryption process converts data to an unintelligible form called *ciphertext*, and the decrypting process converts the *ciphertext* back into its original form, called *plaintext*.

The encryption process goes as follows:

1. The Client generates a symmetric key (K_{AES}) using AES.

2. The Client uses the Local Server's public-key to encrypt the data, the generated symmetric key and a data signature. This data signature is created with the Client's private-key and the Local Server can verify the signature with the Client's public-key. This process proves that the Client had access to its own private-key and is likely to be the one associated with the public-key used.

3. After verifying all information and fetching the configurations from the Main Server, the Local Server encrypts the response using the symmetric key generated by the Client. After receiving the information, the Client can decrypt and process it. This mechanism ensures that if the Client changes, the new receiver can not decrypt a message containing specific configurations of the other one, since it has no access to the symmetric key. On the other hand, the Client can trust the response, since it only shared the symmetric key with the Local Server. Furthermore, symmetric key encryption is faster than asymmetric key encryption, reducing some process time at this stage.

The use of well-known standards provides a secure mechanism for sharing private information and that cannot be tampered by a third party entity. Figure 5.4 resumes this process.

The communication between a Local Server and the Main Server uses HTTPS [ASR99], guaranteeing a secure connection link for sharing configurations.

5.6 Summary

This Chapter details a generic framework for sharing configurations through Near Field Communication. The proposed architecture is composed of three main modules: the Client, the Local Server and the Main Server. The data exchange format allows to have a wide range of configurations and can be extended to other communication technologies. It is also proposed an encryption protocol

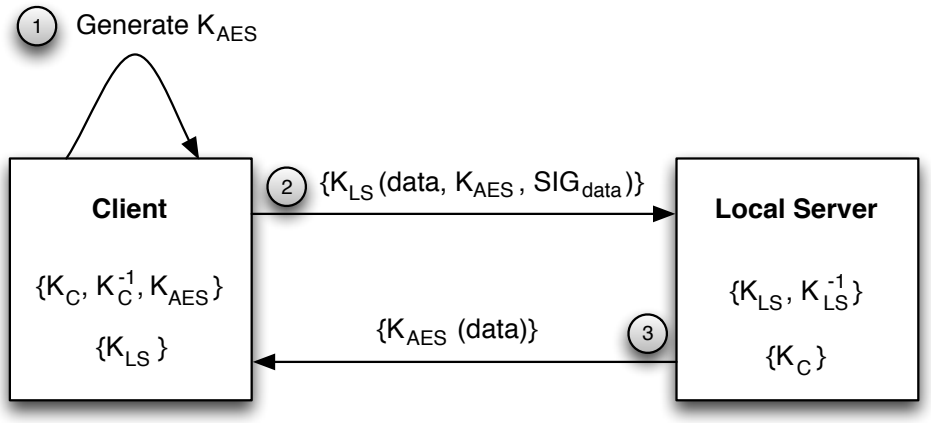

Figure 5.4: Main cryptographic operations between a Client and a Local Server.

for sharing information via NFC, since currently there is no standard defined. The protocol allows safe and transparent connections, safeguarding each party in the process.

Chapter 6

Implementation Details

As described in Chapter 5, the proposed architecture is composed of three main modules: the Client, the Local Server and the Main Server. This Chapter details the developed prototypes for each module, following the functional and non-functional requirements defined for each one. The Main Server is a Ruby on Rails web platform for managing the configurations and that provides an application programming interface (API) with an authentication method. The Client is an Android application that interacts with two different Local Servers and that can be extended to other communication technologies. For the Local Server, there are two different developed solutions - with a common core - that communicate with the Main Server and the Client. Finally, a Local Server's prototype using a proprietary technology of Fraunhofer Portugal was also developed. This Chapter also describes the developed application that use the service to get configurations and functional tests performed with the prototypes.

6.1 Main server

The Main Server is a web platform for managing all configurations and users. It was developed using Ruby on Rails and offers an API for fetching configurations. Ruby on Rails [Rai13] is an open-source web framework for the Ruby programming language and that implements the Model-View-Controller principle. It allows the creation of pages and applications that gather information from the web server, talk to or query the database, and render templates out of the box. This Section describes the prototype main functionalities and design decisions. The following functional requirements were taken into account:

- Add, remove, edit configurations from the database [4.1.2];
- Add, remove, edit users from the database [4.1.2];
- Add, remove, edit configuration values from the database [4.1.2];

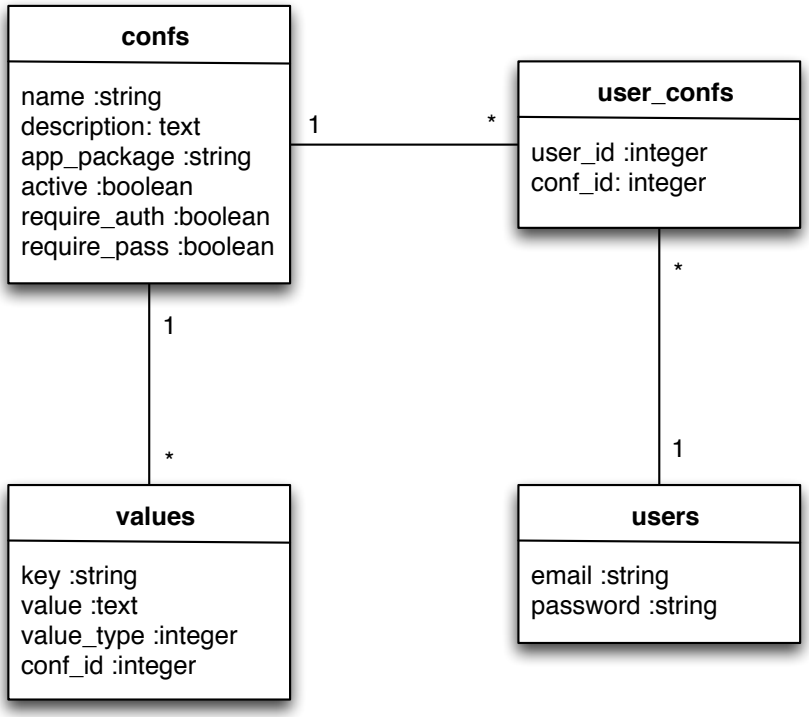

Figure 6.1: Final schema of the the main server's database. Each configuration can have multiple values and users associated. Each user can be associated to multiple configuration.

- Set the users that can access a specific configuration [4.1.2];
- Set current activated configurations with specific authentication methods [4.1.2];
- Get current active configurations for a specific POST request and create a JSON response [4.1.2].

6.1.1 Overview

The Main Server offers a clean and functional web interface for managing configurations. The database design implements the one specified in Section 5.1. The final database scheme is represented in Figure 6.1.

Table `confs` represents the configurations, table `users` the users and table `values` the configuration values. Finally, table `user_confs` allows mapping a configurations to zero or more users, and vice-versa.

Each table has an associated Model, specifying access rules, relationships and all attribute related validations. This customization also defines an application's business logic. The models were created as follows:

The `Conf` model (A.1) can have zero or more associated values and users. The user association is provided by a third table - `user_confs`. In order to create a new configuration, it is necessary to provide a description, a name and the Android application package that will receive

that configuration. This information must be confirmed with the third party application, in order to avoid misleading configuration processes. By default, a new configuration is inactive.

The `User` model (A.2) is characterized by an email and a password, and can be created without password. He can be associated with zero or more configurations. Once more, this association is created using the `user_confs` table.

A `Value` entry (A.3) is always associated with a configuration, belonging to it. In order to create a new value, the user must provide a key, a value type and a value. For a particular configuration, each key must be unique and there are two reserved keys: `id` and `appPackage`, that represent the application identifier and application package of the Android application in the JSON response. As defined in Section 5.4, the service provides the following types: string, integer, float, array of strings, array of integers, array of floats, and URL. Table 6.1 shows an example for each value type.

Table 6.1: The Main Server provides different value types. This table shows an example for each one.

Value Type	Example Key	Example Value
String	ssid	FhP-AICOS
Integer	timeout	10
Float	pi	3.14
Array of Strings	names	[John, Rose, Roy, Oliver]
Array of Integers	points	[4, 13, 3, 8]
Array of Floats	marks	[12.3, 11.0, 17.4, 19.2]
URL	rssFeed	http://feeds.feedburner.com/PublicoRSS

Finally, the `UserConf` model (A.4) maps a user to a configuration. An instance of this object is created when a user is associated with a specific configuration.

6.1.2 Handling configuration requests

The Main Server provides a REST interface to fetch configurations. The `get_config` method handles POST requests from a Local Server and replies with a JSON response after checking for valid configurations based on the information received. Two parameters can be sent along a request: an email and a password. Both are optional, but if the password parameter is set, the email must be too. After receiving a new request, the method iterates through all configurations, in order to select the ones that can be sent. Figure 6.2 represents a flow diagram of the choosing algorithm for each configuration.

The `get_config` function iterates through all configurations and calls `valid_config` for the admissible ones, that creates a JSON representation of a particular configuration. After this iteration, the method renders a JSON response with a success HTTP status. Every response is represented by a JSON Array with zero or more JSON objects. Each object represents a configuration and can contain different key/value pairs. In this example, the response only contains a configuration named `Wifi` that has three values - two strings and one integer:

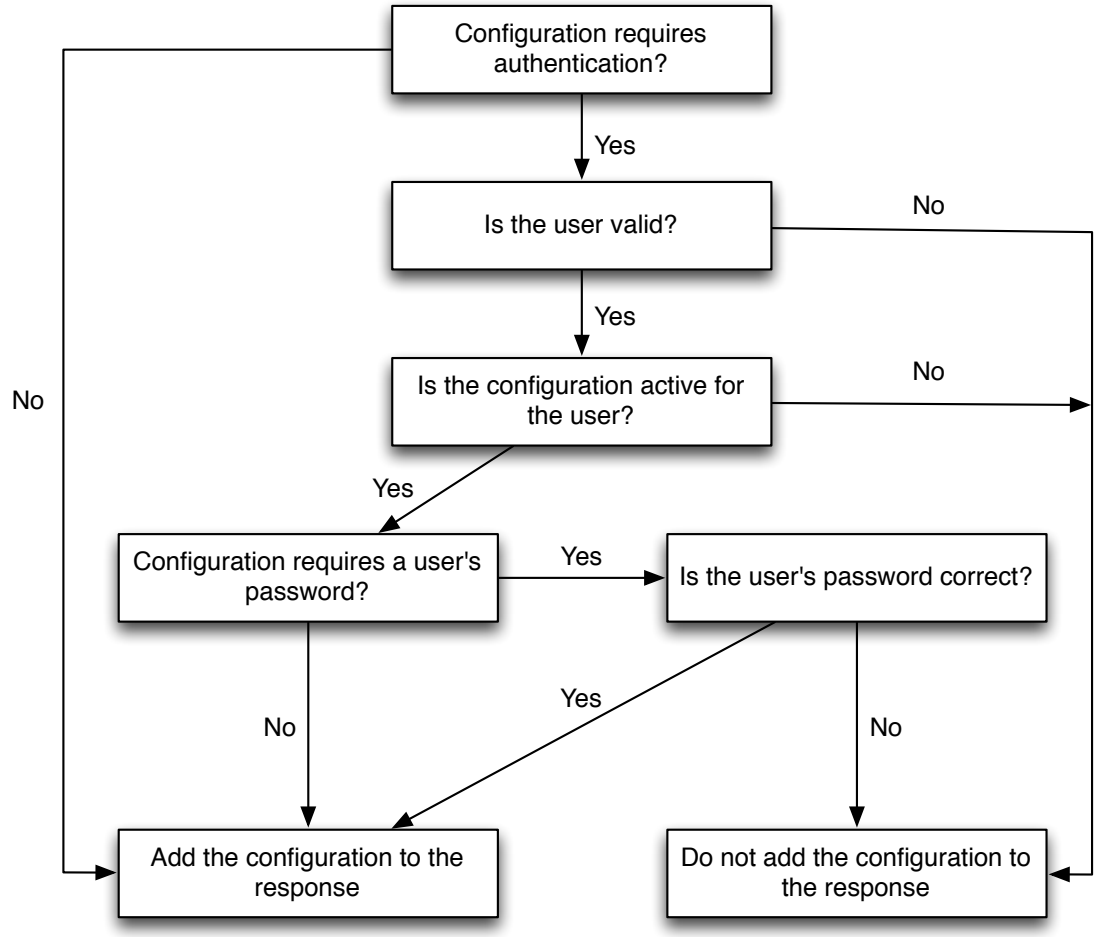

Figure 6.2: Graphical representation of the algorithm responsible for selecting valid configurations for a specific request.

```
1 ["{"id":"Wifi","appPackage":"pt.fraunhofer.nfc.wificonfigurator","ssid":{"value":"
    FhP-AICOS,"value_type":0},"eapMode":{"value":"PEAP","value_type":0},"timeout":{
    "value":2,"value_type":1}}"]
```

6.1.3 Interfaces

This Section describes the most important pages of the developed prototype: main page and configuration page.

6.1.3.1 Main page

In the main page it is possible to quickly select the active configurations at any moment. Regarding authentication methods, if `Require Auth` option is active, only users with access to the configuration will receive it. Furthermore, if `Require Pass` is selected, then it will be also necessary to provide a correct password. If none of these two options are selected, the configuration will be

sent in every request. In this page it is also possible to edit, remove or show each configuration in detail. Finally, a link to add new configurations and manage the users is also provided. Figure B.1 depicts the described interface.

6.1.3.2 Configuration Page

As previously mentioned, each configuration can have zero or more values and users. The configuration page gathers this information and provides an interface to manage it - Figure B.2. The page starts by showing the configuration information and all associated values are listed, allowing to destroy or edit them. Next, it is possible to select which users have access to this particular configuration. A search field is provided in order to easily find a specific user. At the end of the page, a form allows the addition of new values by providing a key, a value and choosing a value type from a pre-defined list.

6.2 Client

The Client, which symbolizes the device being configured, is an Android application that runs on version 4.0 or higher. This Section introduces the Android operating system, details the Generic Client architecture implemented, the configuration process after receiving a valid configuration response and, finally, how it handles all configuration requests through NFC.

6.2.1 Android

Android is a Linux-based operating system for mobile devices. It is developed by the Open Handset Alliance, led by Google. Android is not considered a distribution of Linux, because it implements its own libraries, shells editors, GUIs, programming frameworks and APIs. To run the applications on Android devices, the operating system uses the Dalvilk virtual machine that converts Java bytecodes to dex-code. It was developed to run in devices with low memory requirements [Goo13a].

At the end of 2012, Android had a worldwide smartphone market share of 75%, with more than 500 million devices activated [Goo13b]. Most of these devices don't have an NFC chip, although this tendency is changing. Furthermore, Android is the only mobile operating system with a strong presence on the market that supports NFC [Wik13].

6.2.1.1 Broadcast Receiver

A `BroadcastReceiver` is a base class for code that will receive intents sent by `sendBroadcast()`. An intent is an abstract description of an operation to be performed by the system. It is possible to send and receive broadcasts across applications. All registered receivers for an event will be notified by the Android runtime once this event happens. For instance, an application can register for the `ACTION_BATTERY_LOW` system event which is fired when the battery gets low.

There are two possible classes of broadcasts:

- Normal broadcasts: are completely asynchronous and all registered receiver are run in an undefined order.
- Ordered broadcasts: are delivered to one registered receiver at a time. The system specifies a running order, and each receiver executes in turn, propagating its result to the next receiver. Along the process, a running application can completely abort the broadcast so that it won't be passed to other receivers.

When using `sendBroadcast()`, normally any other application can receive these broadcasts. However, it is possible to safely restrict the broadcast to a single application. Since an application package must be unique, the system will deliver the broadcast only to the wanted application.

An application executing a `BroadcastReceiver` is considered to be a foreground process and will be kept running by the system, except under cases of extreme memory pressure.

6.2.1.2 NFC API

The main format to read and write data on tags is the NFC Data Exchange Format (NDEF) [Goo13d]. Android provides support in parsing the received messages and delivering it as an `NdefMessage` when possible. If the read tag doesn't contain NDEF data or cannot be mapped to a MIME type or URI, it is necessary to handle all the communication with a custom protocol - in raw bytes.

Android also provides a way of pushing an NDEF message onto another device by tapping the devices together. This feature is called Android Beam and is available through a set of NFC APIs, so any application can transmit information.

When NFC is active, the mobile phone is always looking for NFC tags. When it finds one, a special tag dispatch system is called. This system analyzes the tag, parses it, and tries to locate applications that are interested in the scanned data. The process contemplates three steps:

1. The NFC tag is parsed in order to find out the MIME type or a URI that identifies the data payload in the tag;
2. The MIME type or URI and the payload is encapsulated into an intent - an intent is an abstract description of an operation to be performed, in this case, it is referred as a way of starting an activity;
3. Based on the created intent, an activity is started.

After creating an intent with the tag information encapsulated on it, the tag dispatch system sends it to an application that registered its interest on this kind of tags. If more than one application can receive the intent, the user must manually select the correct one. The tag dispatch system defines three types of intents, in order of priority:

1. ACTION_NDEF_DISCOVERED: this intent is used when the type of the tag is recognized, and the tag dispatch system tries to start an Activity with it before any other intent, whenever possible;

2. ACTION_TECH_DISCOVERED: if the ACTION_NDEF_DISCOVERED intent is not handled by an activity, the tag dispatch system tries to start an application with this intent. The intent is also directly started if the information on the tag cannot be mapped to a MIME type or URI, or if the tag does not contain NDEF but is of a known tag technology;

3. ACTION_TAG_DISCOVERED: if there is no activity to handle the previous described intents, the intent is simply started.

Figure 6.3 resumes the tag dispatch system workflow. The Android NFC API is compliant with the NFC Forum standard.

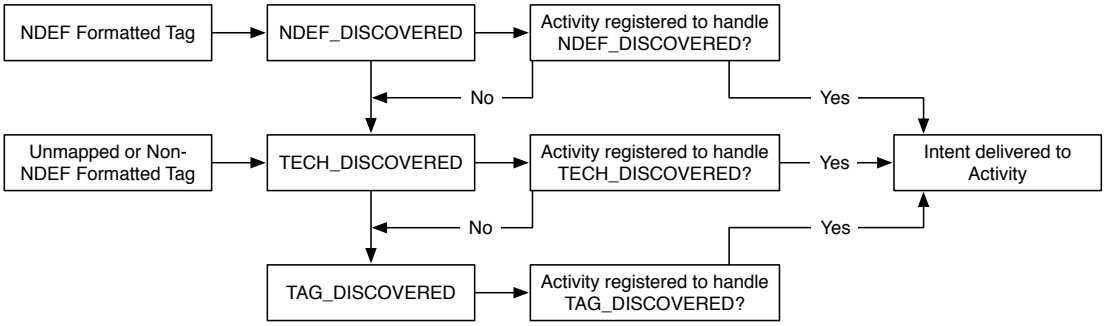

Figure 6.3: Tag dispatch system [Goo13d].

6.2.2 Generic client

The `GenericClient` is a Java class that implements the core functionalities of the Client and that can be used to further extend the system by adding, for instance, new communication technologies.

The class constructor receives a login and a password field, which are very important to build the configuration request that can be fetched at any time. If they are not provided, a default authentication data will be sent.

The `prepareAuthenticationMessage()` returns a byte array with all the necessary information to initiate a configuration process. These method starts by building an empty JSON object. Then, a Cipher object is created by calling the Cipher's `getInstance` method and passing the requested transformation to it. The Client's cipher uses the `RSA` encryption algorithm, with the `ECB` block chaining mode and a `PKCS1Padding` padding scheme. The Local Server's public key allows the initialization of this cipher in encryption mode and later the encryption of the authentication data:

```
1 Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
2 cipher.init(Cipher.ENCRYPT_MODE, localServerPublicKey);
```

A digital signature of the authentication data is also generated. Thus, a new instance of `Signature` that uses the `SHA1withRSA` encryption algorithm is created and initialized with the Client's private key, preparing it for signing:

```
1 Signature signature = Signature.getInstance("SHA1withRSA");
2 signature.initSign(clientPrivateKey);
```

After signing the authentication data and encrypting the signature with the Local Server's public-key, a one-time symmetric key is also generated. This symmetric key is based on `AES` with a 128 bit key. It will be later used to decrypt the Local Server's response:

```
1 // Get the KeyGenerator
2 KeyGenerator kgen = KeyGenerator.getInstance("AES");
3 kgen.init(128);
4
5 // Generate the secret key specs.
6 SecretKey skey = kgen.generateKey();
7 byte[] symmetricKey = skey.getEncoded();
```

The authentication data, the digital signature and the symmetric key are finally added to the JSON object, that is converted and returned as a byte array.

This class also has a function that processes a raw response from a Local Server. It starts by checking if there is any configuration, and if true, the previous generated symmetric key is used to decrypt the information and restore the JSON array sent by the Local Server. For each valid configuration, a `Configuration` object is created. This class processes a JSON object and creates an instance that can be later used to read the properties of a configuration. An array list of configurations is then created and returned.

6.2.3 Configuration Process

After receiving and decrypting a configuration response from the Local Server, it is necessary to broadcast this information to the right applications. The Client application iterates through all `Configuration` objects, creates an intent for each one an sets the broadcast action to the specified application package. Then, it puts all configuration parameters in a `Bundle` object that is sent in the broadcast intent. The `Bundle` class allows the mapping of `String` values to various `Parcelable` types. Finally, it sends the broadcast:

```
1 Intent broadcast = new Intent();
2 broadcast.setAction(configurations.get(i).getAppPackage());
3
4 Bundle bundle = new Bundle();
5 ...
```

```
6 broadcast.putExtras(bundle);
7 sendBroadcast(broadcast);
```

As stated before, Android has a mechanism where an application can subscribe an intent to receive a notification when an event happens. In order to do this, the third party application must declare this intent in its `AndroidManifest.xml` file. This file includes essential information about the application and it must be available to the system, that processes it before running any of the application's code. Here is an example:

```
1 <receiver
2   android:name="pt.fraunhofer.nfc.wificonfigurator.WifiConfigurator"
3   android:exported="true" >
4   <intent-filter>
5     <action android:name="pt.fraunhofer.nfc.wificonfigurator" />
6   </intent-filter>
7 </receiver>
```

The `android:name` attribute shall represent the class that will handle all received broadcasts and the `android:exported` attribute must be set to `true`, so it can receive messages from sources outside its application. The `intent-filter` tag specifies the types of intents that the broadcast receiver can respond to. In this case, it responds to every action set for a specific name. The `android:name` attribute must match the one defined for the application package in the Main Server and that is used by the Client to send the broadcast.

After declaring a receiver in the manifest file, the specified class for handling the broadcasts must extend the `BroadcastReceiver` class and implement the `void onReceive(Context context, Intent intent)` method, that will be called every time a new event is triggered by the Client application. The configuration data can be accessed by calling `intent.getExtras()`, that returns a `Bundle` object containing all required data. In this stage, a third party application should know what parameters to expect and how to handle them. This process is completely agnostic to the Client application.

6.2.4 Handling Configuration Requests through NFC

In order to use the Android NFC API, the Client application instantiates an `NfcAdapter` object, that is obtained by calling `NfcAdapter.getDefaultAdapter(this)`. It also implements and registers in the adapter a `CreateNdefMessageCallback` class, that allows sending messages when there is a new communication link through NFC. Therefore, when a user's device comes close to a Local Server and the user touches the device's screen to allow the operation, the system calls `createNdefMessage(NfcEvent event)`, that returns an `NdefMessage` containing the required information for starting a configuration request.

After starting a configuration process, the Client application uses its package name to subscribe all `ACTION_NDEF_DISCOVERED` tags and enables the foreground dispatch system to the given

`Activity` by using `mNfcAdapter.enableForegroundDispatch()`, that gives priority to the foreground activity when dispatching a discovered tag to an application.

The Android NFC API has some limitations that constrained the development of the prototype:

- Only one `NdefMessage` can be sent per communication link. If we consider two NFC-enabled Android devices, when they are brought together, the NFC dispatch system will look for a callback in both devices, but only the device in which a user touches the screen first will be able to send its data. Furthermore, a user needs to disrupt the current communication link, by moving the devices away, so a new one can be started. Therefore, it is not possible, by default, to establish a bidirectional communication stream;
- When two active devices come close to each other, there must be a physical interaction with one of the device's screen to send information through NFC. Is not possible to surpass this process by disabling it and sending automatically an `NdefMessage`. This verification is important when two devices have an `NdefMessage` to send, because only one can effectively transmit it. However, if one of the devices does not have anything to send, this process will have to also happen.

6.2.5 Interfaces

The Client application is composed of two main activities. When a user opens it and brings his device closer to a Local Server, he will see a message saying to *Touch to Authenticate*. This touch will initiate the configuration process. After receiving, processing the response and broadcasting the configurations, the application closes automatically. If he wants to set his account data, there is a button called *Change Account* that redirects to the second activity. There, he can either enter his login and password or choose a Google account for authentication. The *Save Credentials* button saves the information and redirects him back to the main activity. Figure 6.4 shows these two activities.

6.3 Local Server

Two different prototypes were developed in order to achieve a working solution that could successfully fulfill all use cases. As one of the objective was to develop a solution that could be extended without much effort, a common code base was created. This Section details its implementation and the two developed prototypes.

6.3.1 Generic Local Server

The `GenericLocalServer` is a Java class that implements all core methods without establishing a connection link between the Local Server and a Client. This allows the creation of new solutions with different communication links without reimplementing the communication with the Main Server and the encryption protocol.

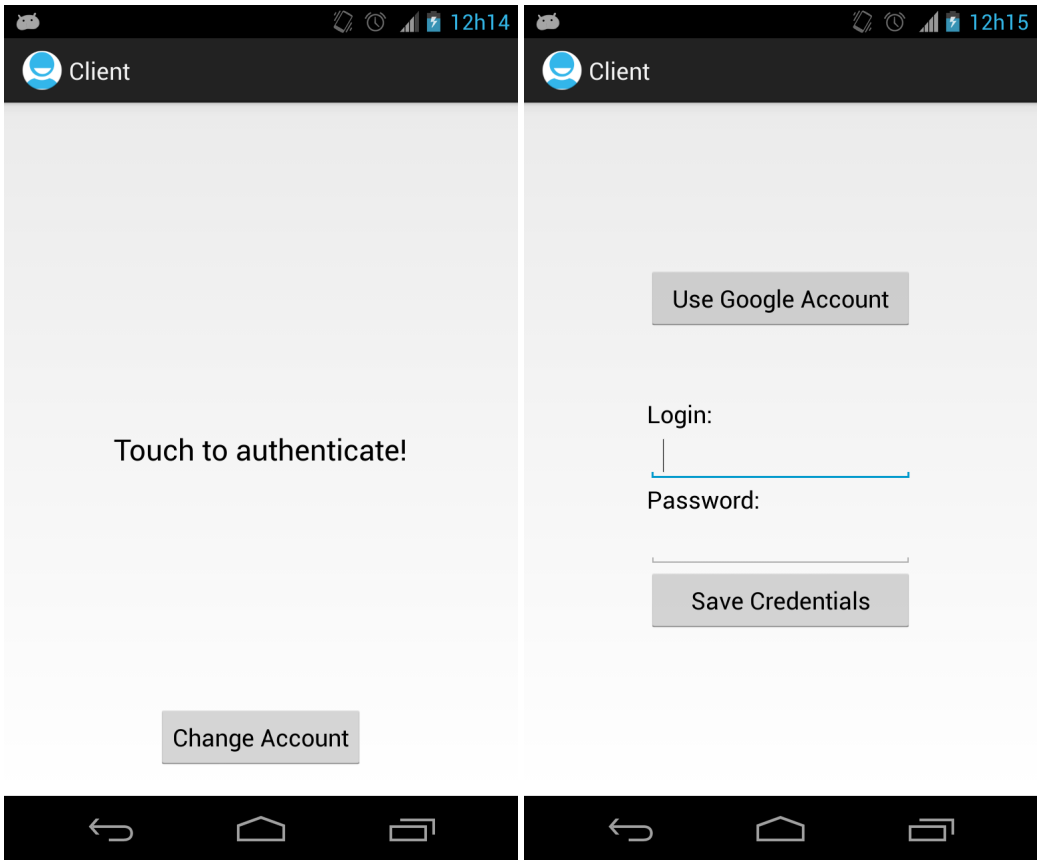

Figure 6.4: Client application activities. The left one represents the launch activity, and on the right one a user can change his authentication data.

`GenericLocalServer`'s constructor receives a byte array that represents the information received by a Client. It starts by creating a JSONObject and pulling out the authentication information, the symmetric key and the data signature. Then, it verifies if the message came from a reliable Client by checking the data signature with the Client's public-key. If this verification is positive, the authentication data and the symmetric key will be decrypted using the Local Server's private-key and saved in the object.

The `getConfigurations()` method retrieves all valid configurations from the Main Server by creating a new HTTPS request with the authentication data provided. After receiving a response from the Main Server, the message is encrypted using the previously received symmetric key and returned in a byte array. Then, the Local Server can send back a response to the Client.

6.3.2 ACR122U NFC Reader

The ACS ACR122U NFC Reader[Advb], represented in Figure 6.5, is a contactless smart card reader and writer that operates at 13.56 MHz. It is a plug-and-play USB device and provides an access speed of up to 424 kbps. The proximity operating distance of ACR122U is up to 5 cm, depending on the type of contactless tag or device in use. Internally, it has a PN532 NFC Controller chip that allows the establishment of NFC connections.

Figure 6.5: ACS ACR122U NFC Reader, a contactless smart card reader[1].

Although it does not provide a native library for sending and receiving messages with an Android device, a library existed - `ismb-snep-java`[2] - that seemed to provide an implementation for the communication link. As described in Section 6.2.1.2, Android is compliant with the NFC Forum standard, so the objective of this library is to implement the Simple NDEF Exchange Protocol (SNEP) that allows sending NDEF Messages between the reader and an Android device. However, and after performing some tests with a Nexus 4, the reader was not able to receive messages from the device, although it could send a pre-defined message. Thus, it was necessary to rewrite most of the library to send and receive NDEF Messages with variable length. The following Section details the major implementation details of the protocols needed to share messages between an ACR122U reader and an NFC Forum device.

6.3.2.1 Initial Setup

The Java Smart Card I/O API[3] defines an API for communication with Smart Cards and that works with the ACR122U reader. Thus, it was used to communicate with the PN532 NFC chip of the reader. This API provides methods to connect and send application protocol data unit (APDU) commands to a reader.

The ACR122U does not provide a direct link to a smartcard, for example. All commands must be sent to the reader itself, and then handled by it. Hence, the reader's API defines these commands as *pseudo* APDUs [Adva]. They permit access to the functionality of the PN532 builtin chip, or

[1] http://cdn.shopify.com/s/files/1/0109/3792/products/ACR122U_1024x1024.jpg?25
[2] https://code.google.com/p/ismb-snep-java/
[3] http://docs.oracle.com/javase/6/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html

some reader specific functionality, like changing the LED status. For this prototype, the focus was on the direct command APDUs, considering that it was necessary to send direct messages to the internal NFC chip. Thus, the header for sending commands is:

`0xFF 0x00 0x00 0x00 0xii`

Where `0xii` indicates the number of bytes to send after the header. This sequence must be present in all commands meant for the NFC-enabled device. Since this header has to be present in every command, hereinafter it is omitted.

In order to receive or send messages, the first step is to initialize the reader as target, so it can wait for an initiator to start the communication. Table 6.2 represents the available command to configure the PN532 as target [NXP07].

Table 6.2: Command to configure the PN532 as target.

D4	8C	Mode	MifareParams	FeliCaParams	NFCID3t	LEN Gt	Gt	LEN Tk	Tk

`Mode` byte indicates which mode the chip should respect and it is different for sending and receiving messages. `MifareParams`, `FeliCaParams` and `NFCID3t` bytes contain the information needed to activate the reader in different speeds and follow the default standards of the API. `LEN Gt` field codes the number of general bytes, and it is mandatory. The `Gt` field has a variable size (max. 47 bytes) and contains the general bytes. The Logical Link Control Protocol (LLCP) , that delineates the procedural means for the transfer of upper layer information units between two NFC Forum Devices, specifies the use of a three octet sequence `0x46 0x66 0x6D` as the *NFC Forum LLCP magic number* and that is encoded into the general bytes field [NFC11a]. Finally `LEN Tk` codes the number of historical bytes (`Tk`). This information is optional and not used in the implementation.

After this initial setup, the NFC reader must indicate if it wants to receive or send messages.

6.3.2.2 Sending Messages

`PassiveOnly` flag is used to configure the `Mode` field when initiating the reader as target. This means that the reader will only work in passive mode, i.e. it will refuse an active communication mode. This working mode simulates the mechanism of an NFC tag.

After initing the reader as target, it will receive a request from the initiator device asking for information about how the reader will operate. Considering that we want to send messages, a connection APDU response is used to establish a data link connection between the reader and a device, as defined in the LLCP (Table 6.3). The reader API specifies a command (`SetData`) that can be used to supply the PN532 with data that it wants to send back to the initiator (`0xD4 0x8E`) and that precedes all LLCP messages intended to inform the NFC Forum device.

Each LLC APDU contains two address fields: the Destination Service Access Point (`DSAP`) and the Source Service Access Point (`SSAP`). In this case, the `DSAP` always represents an NFC Forum device, so the NFC Forum Simple NDEF Exchange Protocol (SNEP) address is used

Table 6.3: Format of the connect APDU.

DSAP	PTYPE	SSAP	Information
DDDDDD	0100	SSSSSS	Parameter List

(000100) [NFC11b] . The SSAP field assigns an address for an upper layer service request that enables SNEP support (100000). This field is used in all requests and responses. The parameter list is optional and ignored.

After receiving a positive acknowledgment from the device, it is possible to send information. As specified before, the reader will send NDEF messages, so it will use the SNEP [NFC11b] over the LLCP [NFC11a] established link. Table 6.4 shows the format of an Information APDU, that will contain the NDEF Message.

Table 6.4: Format of the information APDU.

DSAP	PTYPE	SSAP	N(S)	N(R)	Information
000100	1100	100000	NNNN	RRRR	Service Data Unit

The N(S) field indicates the sequence number associated with this APDU and the N(R) indicates that Information APDUs numbered up through N(R) - 1 have been received by the initiator. The Information field will contain all necessary information to send an NDEF Message via SNEP. The prototype default values are detailed in Table 6.5.

Table 6.5: SNEP put request message format.

Version	Request	Length	Information
0x10	0x02	4 bytes	NDEF Message

The first octet indicates the format of the SNEP message and the sender's capacity for understanding further SNEP communication. The most significant 4 bits of the Version field denote the major protocol version and the least significant 4 bits denote the minor protocol version. The request field represents the Put command, asking the device to accept the NDEF message transmitted with the request. The Length field represents the size of the Information field. Finally, the Information field will contain an NDEF Message [NFC06] that can contain one or more NDEF records. The record layout is represented in Table 6.6.

The Message Begin (MB) field indicates the start of the message and the Message End (ME) field its end. The Chunk Flag (CF) field tells if this is either the first record chunk or a middle record chunk of a chunked payload. The Short Record (SR) field, if set, defines that the Payload Len is a single octet. The IL field is set to one if the ID Len is present in the header. The Type Name Format (TNF) indicates the structure of the value of the Type field. The Type Len field is a byte that specifies the length of the Type field. For certain values of the TNF it is zero. The Payload Len specifies the payload length, and its field size is determined by the value of the SR bit. The Type field is an identifier that describes the type of the payload, following the structure, encoding and format implied by the TNF value. The ID is optional and is an identifier in the form

Table 6.6: NDEF record layout.

MB	ME	CF	SR	IL	TNF
1 bit	1 bit	1 bit	1 bit	1 bit	3 bits
Type Len	**Payload Len**	**ID Len**	**Type**	**ID**	**Payload**
1 byte	1 or 4 bytes	1 byte	receiver	1 byte	message

of a URI reference. Finally, the `Payload` field carries the data of the NDEF record. The format of this data is completely transparent to the NDEF data exchange format.

The first byte of every NDEF message sent by the prototype is the same: `0xC2`. It determines that the message is the first and last record to send, it is the first chunk of the message, the `Payload Len` has 4 bytes, the `ID` field is not present and that the `TNF` format used is the Media-type (MIME). Then, the `Type` field represents the receiver's name that will be notified in the Android device. At last, the configurations are sent in the `Payload` field.

As the ACR122U imposes a limit of 255 bytes per message, a chaining mechanism is necessary when the information exceeds this limit. Both SNEP and LLCP provide a chaining protocol. This solution implements the one defined by the LLCP. In order to use it, the NDEF record is normally created, even though it can exceed 255 bytes. Then, the chaining is performed at a lower layer, where the reader sends multiple information APDU, incrementing the sequence field. The device will use this information to reconstruct the message after receiving all chunks.

These are the principal APDUs necessary to send a message from an ACR122U to an NFC Forum device. After sending a request, a disconnect APDU shall also be sent to the initiator, in order to correctly end the communication.

6.3.2.3 Receiving Messages

`DEPOnly` flag is used to configure the PN532 to accept being initialized only as NFC-DEP target. This working mode follows the NFC Forum standard for the NFC Forum Peer Mode, where the initiator starts the communication, and the target responds to it. In this mode it is possible to accept a request from a device to send a message and reply positively in order to receive it.

Subsequently, the reader will wait for a connection request from an initiator. A connection complete (`CC`) LLCP APDU is sent to confirm that the reader wants to receive the message. Table 6.7 shows the format of this message. Note that this command is preceded by the ACR122U common header and the `SetData` header.

Table 6.7: Format of the connection complete APDU.

DSAP	PTYPE	SSAP	Information
100000	0110	000100	0x05 0x01 0x01

Contrary to what happens when sending messages, the `DSAP` field represents the reader, whereas the `SSAP` field the device. The `PTYPE` indicates that this APDU acknowledges the device and accepts the Connect command. This Connection Complete APDU (`CC`) also specifies a value

for the parameter list. The `0x05 0x01 0x0b` indicates the Receive Window Size (`RW`). A receive window size of one indicates that the reader will acknowledge every Information APDU before accepting additional Information APDUs.

In order to receive the actual message, a `GetData` (`0xD4 0x86`) request is sent to the PN532. Every time a new Information APDU is read, it is necessary to send a LLCP Receive Ready (`RR`) APDU. This command has the format shown in Table 6.8.

Table 6.8: Format of the receive ready APDU.

DSAP	PTYPE	SSAP	Sequence
100000	1101	000100	0000SSSS

The receive sequence number (`N(R)`) indicates that Information APDUs numbered up through `N(R) - 1` have been correctly received by the reader. This process is repeated as long as the reader receives new chunks of the final message. When the length of the message indicates that it is the last chunk, the next message will contain a Disconnect Mode (`DM`) APDU. In order to correctly terminate the connection, the reader also has to send a `DM` APDU. Table 6.9 exemplifies this command.

Table 6.9: Format of the disconnect mode APDU.

DSAP	PTYPE	SSAP	Reason
100000	1101	000100	0x00

The `Reason` field indicates that the LLC has received a `DM` APDU and is now logically disconnected from the data link connection. After disconnecting, it is possible to join all chunks of the message, remove the service bytes and create an NDEF Message from the remaining information.

6.3.2.4 Discussion

Although a fully workable prototype was developed, the performed tests with the ACR122U are not very satisfactory. The reader has a LED that indicates the presence of an NFC-enabled device. This works if the correct drivers are installed and it is a factory default functionality. When testing with a LG Nexus 4, that has a Broadcom NFC chip[4], the reader was able to always recognize it and establish a communication link. However, with a Samsung Galaxy Nexus, a Samsung Nexus S and a HTC One X (they all carry an NXP NFC chip[5]) the reader did not recognize the devices in 95% of the tests. The devices would have to be fully stable and in a specific position in order to establish the link. Even though this could indicate a faulty hardware, the results with a Nexus 4 dispelled this doubt.

Furthermore, the error recovery mechanism of the reader cannot prevent some crashes. If a device was to be removed from the reader while they were exchanging information, the reader would crash, being unable to answer or receiving any APDUs. It would only work after reconnecting it

[4]http://andytags.com/nfc-tags-nexus-4–10-compatibility.html#.UbCUWPZxuJM
[5]http://www.yunnfc.com/a-comprehensive-list-of-nfc-enabled-phones/

to an USB port. Additionally, the communication speed with the reader using the Java Smart Card I/O API was not the expected, delaying the protocol establishment and, consequently, the time to receive and send information between the reader and a device.

Considering the obtained results with the ACR122U, a new prototype was developed. The first approach used an Android NFC-enabled device as Local Server. However, this solution had two major problems: it was necessary to move apart the devices after the authentication handover and then bring them together again to send the configurations and it was obligatory to touch the Local Server's screen to authorize the NFC connection. This solution had bad usability and required an extra unwanted effort to establish a new communication link. The use of an Android device would be achievable if it was possible to remove these two constraints. The following section explains how it was done.

6.3.3 Android Custom Rom

The Android Open Source Project allows the creation of custom variants of the Android software stack[6]. It is possible to download the source code, make the changes we want and then build it for a specific device. Thus, Android 4.1.2 was downloaded from the source code repository, customized and built for the Samsung Nexus S.

The Android Local Server's application uses the Android NFC API and the Generic Local Server class to provide a complete bridge between a device and the Main Server. Furthermore, the developed application was included as a system application of the Android operating system. A system application can access restricted libraries of the Android system. As aforementioned, one of the original problems was the need to move apart the devices so it would be possible to disconnect the current communication link and create a new one. There is no official API for requesting a new message. Thus, a `sendNewMessage` method was introduced in the `NfcAdapterService` class, and allows resetting the NFC communication link and start a new connection:

```
1 @Override
2 public boolean sendNewMessage() throws RemoteException {
3   mP2pLinkManager.onLlcpActivated();
4   return true;
5 }
```

Furthermore, the `onLlcpActivated()` method of the `P2pEventListener` class was also modified:

```
1 public void onLlcpActivated() {
2   ...
3   switch (mLinkState) {
4   ...
```

[6] http://source.android.com/

```
5   case LINK_STATE_UP:
6     //deactivates current link
7     onLlcpDeactivated();
8
9     //creates a new link
10    mLinkState = LINK_STATE_UP;
11    mSendState = SEND_STATE_NOTHING_TO_SEND;
12    if (DBG) Log.d(TAG, "onP2pInRange()");
13    mEventListener.onP2pInRange();
14
15    prepareMessageToSend();
16    if (mMessageToSend != null || (mUrisToSend != null && mHandoverManager.
          isHandoverSupported())) {
17      mSendState = SEND_STATE_NEED_CONFIRMATION;
18      if (DBG) Log.d(TAG, "onP2pSendConfirmationRequested()");
19      mEventListener.onP2pSendConfirmationRequested();
20    }
21    break;
22    ...
23  }
```

Normally, if there was already a communication link (`LINK_STATE_UP`) this method would only send a debug message and return. In this solution, it ends the current link and creates a new one.

6.3.3.1 Overcoming the Need to Touch the Screen

Besides introducing a programatic way of creating a new NFC communication link, there was still the need of overcoming the "Touch to Beam" functionality, that requires a touch in the screen to allow an NFC link establishment.

The `P2pEventManager` class handles all NFC events between two active devices. When a new interaction is created, a new instance of the class is instantiated. The class constructor creates a `SendUi` class object, that is responsible for handling the UI animation around Android Beam. When the user touches the screen, a callback function that belongs to the `P2pEventManager` class is called (`onSendConfirmed()`). So, instead of creating a new instance of `SendUi`, the constructor calls `onSendConfirmed()`, automatically invoking the `createNdefMessage` method of the foreground activity - in this case it will be the Local Server's application.

This change could introduce a new problem: the Local Server's device will always call the foreground activity handler each time an NFC-enabled device comes closer to it. For instance, if a device wanted to send an NDEF Message but the Local Server's device also had a foreground activity extending the `CreateNdefMessageCallback` class, it would be practically impossible to touch the screen first at the user's device. However, this problem does not occur with the developed prototype. As referred before, the operation always starts at the user's device, where he has to touch the screen to allow the configuration process and send his authentication data.

Since the Local Server's phone will not have any opened application, there will be no foreground application with an NFC callback, and the user can start the process without problem. On the other hand, when the Local Server receives the configuration request, it will query the Main Server and then automatically send the configurations through NFC, as explained before.

It is important to state that the user's device does not need to have a custom operating system (OS) image. Furthermore, the Local Server's device is provided by the service, so there is no constrain on changing its OS image. The Local Server's does not impose any limitation to the system, and it was tested with different NFC-enabled phones (Nexus 4, Samsung Nexus S, Samsung Galaxy Nexus) with a success rate of 100%, mostly because it uses the Android NFC API to communicate in a lower level.

6.4 ULF-MC Helmholtz Coils

Although the focus of this dissertation is in the use of Near Field Communication, a second approach using a different communication technology was developed. The main objective is to prove the extendability of the suggested architecture, even though the ULF-MC Helmholtz Coils project introduces heavy constrains.

Ultra Low Frequency Magnetic Field Communication (ULF-MC) is an internal project of Associação Fraunhofer Portugal Research[7]:

> *"Ultra Low Frequency Magnetic Field Communication (ULF-MC) consists in a wireless communication technology for smartphones, based on the generation of low-frequency varying magnetic fields. As most of the smartphones are already equipped with 3 axis hall effect sensors (i.e. electronic compasses) – capable of measuring variations in magnetic fields – these are practically all-set to communicate using this technology. ULF-MC can be used for multiple applications such as Indoor Location, Location-based Services and Mobile Payment. It has also been awarded the Galileo Master prize at the European Satellite Navigation Competition, 2012 edition (highest distinction)."*

The current prototype allows the transmission of messages to an Android device with a bit rate of 40 bits/s. However, the developed algorithm constrains this value by adding service bits, and it is only possible to send about 20 bit/s. This value, compared with the NFC technology, for instance, is very low, and it was necessary to perform some optimizations in order to send a configuration in good time.

This new prototype mimics a Local Server's behavior, but without communicating with the Main Server and without using the encryption mechanism. Currently, the ULF-MC requires a fixed message length, that must be parameterized in the antenna and in the Android device. Since the configuration message has a variable size, it would be possible to retrieve the configurations

[7] http://www.galileo-masters.eu/index.php?anzeige=overall12.html

from the Main Server, but the device would not know its size, thus not receiving it. It was decided to simulate the Main Server interaction by always returning the same type of configuration.

Furthermore, and in order to reduce the size of a configuration message, a six bit encoding was defined:

```
char[][] encodingMatrix = new char[][]
{
  {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p'},
  {'q','r','s','t','u','v','w','x','y','z','A','B','C','D','E','F'},
  {'G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V'},
  {'W','X','Y','Z','1','2','3','4','5','6','7','8','9','0','-',','}
};
```

The first two bits of each character indicate the row and the four remaining indicate the column of the array. This encoding allows saving two bits per byte. For instance, the character `R` is encoding as `101011`.

Two configurations were selected to test the prototype. Both represent a Wi-Fi configuration with different authentication methods. The first one connects the device to a WEP Secured Wireless Network and the other to a Protected EAP Wireless Network.

It was also necessary to abolish the use of JSON to represent a configuration, so it was possible to reduce the size of a configuration to a minimum. Instead, a new format for a wireless configuration was defined and it is represented in Table 6.10.

Table 6.10: Format of a wireless configuration usign the ULF-MC prototype.

Package	Encryption	MAC Address	Password Length	Password
6 bits	2 bits	24 bits	4 bits	[0, 90] bits

The `Package` field can only have one character, the `Encryption` field can represent up to three security authentication methods (WEP, WPA/WPA2 PSK and PEAP), the `MAC Address` represents the 3 most significant hexadecimals of the network unique identifier, and the `Password` field can indicate the network's password, if any. The `Password Length` can express up to 15 characters. Thus, and considering the six-bit encoding mechanism, the `Password` field can have a maximum size of 90 (15x6) bits.

A limit size of 126 bits was defined for every Wi-Fi configuration. After specifying both in the antenna and Client application this size, it is possible to place the phone near the prototype and wait for the configuration process to happen. It can take approximately 8 seconds to configure a wireless configuration.

6.4.1 Using the Library to Send Configurations

The ULF-MC project provides a library to receive information in an Android device by implementing a `ULFMCListener` class. Every time something is received, the `onULFMCEventReady` is

called, allowing to process the received data. In a normal scenario, it would be possible to use the `GenericClient` class to handle the received configuration. However, and considering that it was decided not to use the encryption protocol, it must be processed differently. After decoding the configuration, a JSON object, like the one sent by the Main Server, is created and it is broadcasted to the right application using the same mechanism proposed in Section 6.2.3.

6.5 Test Applications

This Section describes two developed applications that use the developed prototype, proving that it works in different contexts and it does not focus in one type of application.

6.5.1 Wifi Configurator

It uses the prototype to receive configurations and create new Wi-Fi connections. It can connect to networks with different types of security mechanisms and authentication methods. For instance, an airport could provide an easy way to configure Wi-Fi connections through NFC. The user would only need to bring together his mobile phone and a NFC platform - considering that he has valid credentials - to establish the connection.

6.5.2 RSS Reader

A traditional Rich Site Summary (RSS) Reader application for a Smartphone normally needs an Internet connection to retrieve the news from its sources. Using the developed prototype it is possible to provide a different way of doing it. Thus, an alternative application was developed, where the user only needs to bring the Smartphone closer to a Local Server, that can be provided in different scenarios, and have it configured with the latest news. This scenario does not involve a specific configuration handover, but the application could also represent a user's ticket. Thus, it could take advantage of the authentication process to validate the ticket, while receiving the news. Nevertheless, this example also shows that the service could be used to also send information that do not totally represent a configuration.

6.6 Functional Tests

A set of functional tests were performed to determine if the prototypes -using a Nexus S and a ACR122U - returned the excepted output. They tested distinct scenarios, with different accounts and active configurations. Different NFC-enabled mobile phones and Android versions were used. The results show that the renewed Local Server was approximately 2.5 times more faster than the older one, that used an ACR122U reader. The configuration time obtained with the final version was the expected for the operation. The prototype was able to fully configure a Wi-Fi connection in 4 seconds, for example.

Tables 6.11, 6.12, 6.13, 6.14 and 6.15 represent all functional tests performed with the final Local Server, indicating the expected and actual result. All tests indicate that the prototype always returned the expected result, being able to handle different scenarios.

Table 6.11: Functional test 1.

Client Application:

Account	fraunhofer.pt\carlos.babo
Password	*true*
Valid password	*true*

Main Server:

Configuration	Active	Require Auth.	Require Pass.	Active for user
Wi-Fi Configurator	*true*	*true*	*true*	*true*
RSS Reader	*false*	-	-	-

Expected Result: Wi-Fi connection configured with success.
Actual Result: Wi-Fi connection configured with success.
Configuration time (Google Nexus S): 3.9s.
Configuration time (ACR122U): 9.4s.

Table 6.12: Functional test 2.

Client Application:

Account	fraunhofer.pt\carlos.babo
Password	*true*
Valid password	*false*

Main Server:

Configuration	Active	Require Auth.	Require Pass.	Active for user
Wi-Fi Configurator	*true*	*true*	*true*	*true*
RSS Reader	*false*	-	-	-

Expected Result: Nothing configured.
Actual Result: Nothing configured.

6.7 Summary

The Main Server is a Ruby on Rails web platform that provides (i) an interface to manage the configurations and (ii) an API with an authentication method to retrieve configurations.

The Client is an Android application that (i) receives configurations from a service provider, (ii) broadcasts the received configurations to the third party applications and (iii) allows the user to specify different authentication methods.

Finally, three different approaches for a Local Server are detailed, viz. (i) one using an ACR122U NFC Reader, (i) one using a Nexus S device running on a custom operating system image and (iii) one using ULF-MC, a proprietary technology of Associação Fraunhofer Portugal

Table 6.13: Functional test 3.

Client Application:

Account	fraunhofer.pt\carlos.babo
Password	*true*
Valid password	*true*

Main Server:

Configuration	Active	Require Auth.	Require Pass.	Active for user
Wi-Fi Configurator	*true*	*true*	*true*	*true*
RSS Reader	*true*	*false*	*false*	-

Expected Result: Wi-Fi connection and RSS feed application configured with success.
Actual Result: Wi-Fi connection and RSS feed application (news feed with approximately 29000 characters) configured with success.
Configuration Time (Google Nexus S): 12.3s
Configuration Time (ACR122U): 27.8s

Research. The first two prototypes (i) receive requests from the Device, (ii) communicate with the Main Server via HTTPS to fetch the configurations and (iii) and send back the configurations to the Device through a secure NFC channel. The ULF-MC technology proves that the solution presented here can be extended to other wireless interfaces.

Two proof-of-concept Android applications - RSS Reader and Wi-Fi Configurator - show that (i) the solution works in different contexts and (ii) it does not focus in one type of application.

The final prototype, using a Nexus S, underwent different functional tests, proving that it always returns the expected result, being able to handle different configuration scenarios.

Table 6.14: Functional test 4.

Client Application:

Account	carlos.babo@fraunhofer.pt
Password	*false*
Valid password	-

Main Server:

Configuration	Active	Require Auth.	Require Pass.	Active for user
Wi-Fi Configurator	*true*	*true*	*true*	*false*
RSS Reader	*true*	*false*	*false*	-

Expected Result: RSS feed application configured with success.
Actual Result: RSS feed application configured with success (news feed with approximately 29000 characters).
Configuration Time (Google Nexus S): 10.5s
Configuration Time (ACR122U): 25.8s

Table 6.15: Functional test 5.

Client Application:

Account	carlos.babo@fraunhofer.pt
Password	*false*
Valid password	-

Main Server:

Configuration	Active	Require Auth.	Require Pass.	Active for user
Wi-Fi Configurator	*true*	*true*	*true*	*false*
RSS Reader	*true*	*true*	*false*	*false*

Expected Result: Nothing configured.
Actual Result: Nothing configured.

Chapter 7

Conclusion and Future Work

The research conducted in this dissertation led to the development of a generic framework for sharing configurations through Near Field Communication between an Android NFC-enabled device and an external service. With the final prototype it is possible to rapidly and effortlessly configure services on a mobile device. A user only needs to bring his device near a service provider and accept the configuration process, that will automatically start.

Near Field Communication proved to be an excellent technology to achieve a solution with minimal user intervention. The short-range distance ensures that a user is always in control of the operation and that he can cancel it whenever wanted. The setup free process also favors the configuration process, since an unexperienced user will not need to know technical details about how to configure a connection.

The developed service can provide multiple configurations at the same time, for different users, and with distinct authentication methods. This process is completely transparent to the final user and to the third party application owner that wants to use the service to configure his application.

An encryption protocol guarantees security to the process. Although a possible attacker can intercept and save the authentication data or the configurations, he will not be able to decrypt the information, since a public-key cryptographic system secures the authentication transactions and a one-time generated symmetric key secures the configuration data. Furthermore, it is not possible for a device to decrypt a set of configurations destined to another one.

The defined data format for sharing configurations through NFC allows the specification of a countless number of different configurations, with distinct parameters and data types. The URL data type introduces a dynamic way of fetching up to date information from a website, thus allowing a temporal customizable configuration.

The first Local Server prototype, using an ACR122U, did not meet all defined requirements, so a new one had to be developed. The final version, running on a Nexus S with Android 4.1, meets all defined requirements and its able to provide the service without requiring more than one touch. It uses a modified version of the Android NFC stack, that was customized to remove two

major constraints: the *touch to beam* functionality and the need to move apart the devices in order to later initiate a new communication link.

The Wi-Fi Configurator and the RSS Reader applications use the external service and show that it can be used in different scenarios and contexts. These two applications offer new and innovative ways of providing information, without requiring technical knowledge or the use of mobile data connections such as Wi-Fi or 3G.

Thus, the initial proposed objectives were completely achieved. Furthermore, it was possible to extend the service to use the ULF-MC prototype, showing that the proposed solution can work with different communication technologies.

The performed functional tests with the prototype show that the service always returns the expected configurations, meeting all functional and non-functional requirements.

The following list enumerates a summary of all contributions of this dissertation:

1. A specification and an architecture for a generic framework for sharing configurations through NFC between a mobile device and a generic external service;
2. An encryption scheme to guarantee security and authenticity when sending information through NFC;
3. A data format for sharing configurations through NFC;
4. A library that allows sending messages between an NFC Forum Device and an ACR122U reader;
5. An Android custom ROM that removes the need to touch the screen to beam a message through NFC;
6. A custom Android NFC API that provides a method to programmatically eliminate a current NFC link and create a new one;
7. Three prototypes for a Local Server based on the defined architecture using (i) an ACR122U NFC reader, (ii) a custom Android ROM (iii) the Fraunhofer's ULF-MC project;
8. A configuration format for sharing Wi-Fi configurations using the ULF-MC technology;
9. A prototype for a Client based on the defined architecture for the Android OS;
10. A prototype for a Main Server based on the defined architecture in Ruby on Rails;
11. Two proof-of-concept Android applications - RSS Feed and Wi-Fi Configurator - that receive configurations from the service.

Regarding future work, and considering the proposed architecture, it is possible to study other NFC similar communication technologies and extend the Local Server to allow multiple connection types. Client applications can be developed for other operating systems and even other devices than smartphones, as well as develop new use case applications that use the service.

Appendix A

Main Server Models

```
1  class Conf < ActiveRecord::Base
2    attr_accessible :description, :name, :app_package, :active, :require_pass, :
         require_auth
3    validates_presence_of :description, :name, :app_package
4
5    has_many :values
6    has_many :user_confs
7  end
```

Listing A.1: Conf model.

```
1  class User < ActiveRecord::Base
2    attr_accessible :email, :password
3
4    has_many :user_confs
5  end
```

Listing A.2: User model.

```
1  class Value < ActiveRecord::Base
2    belongs_to :conf
3    attr_accessible :key, :value, :value_type
4
5    validates_presence_of :key, :value, :value_type
6    validates_uniqueness_of :key, :scope => :conf_id, :message => "%{value} is
         already in use."
7
8    validates :key, :exclusion => { :in => %w(id appPackage),
9      :message => "%{value} is reserved." }
10
```

```
11 end
```

Listing A.3: Value model.

```
1 class UserConf < ActiveRecord::Base
2   belongs_to :user
3   belongs_to :conf
4   attr_accessible :user_id, :conf_id
5 end
```

Listing A.4: UserConf model.

Appendix B

Main Server Interfaces

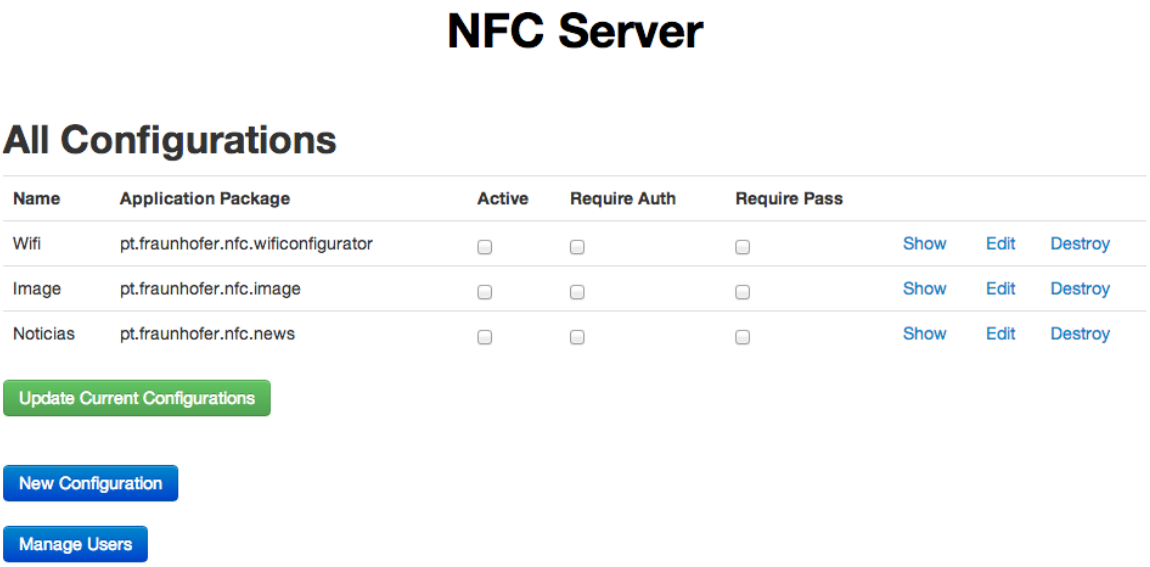

Figure B.1: Main page interface example of the Main Server web application.

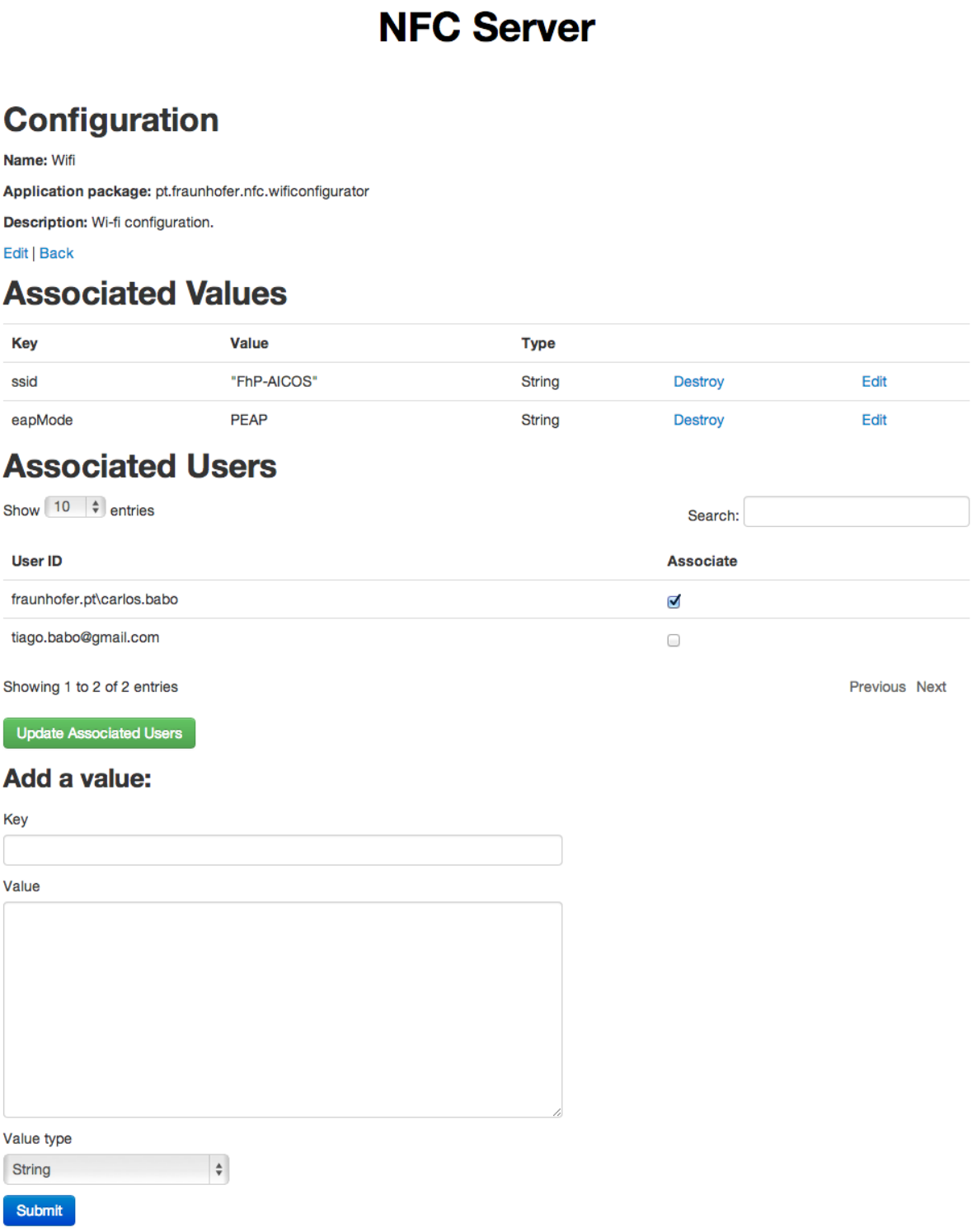

Figure B.2: Configuration page interface of the Main Server web application.

References

[Abe06] MT Abel. Means for provisioning and managing mobile device configuration over a near-field communication link. *US Patent App. 11/354,508*, 2006.

[ABPW07] Y Anokwa, G Borriello, T Pering, and R Want. A user interaction model for NFC enabled applications. In *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007*, 5th Annual IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2007, pages 357–361, Computer Science and Engineering, University of Washington, Seattle, WA 98105, United States, 2007.

[Adva] Advanced Card Systems Ltd. Acr122u nfc reader - application programming interface. Technical report.

[Advb] Advanced Card Systems Ltd. Acr122u nfc reader - technical specification. Technical report.

[All06] Open Mobile Alliance. Nfc data exchange format (ndef) - technical specification. Technical Report July, 2006.

[All08] Open Mobile Alliance. OMA Device Management Protocol. pages 1–53, 2008.

[All13] Wi-Fi Alliance. Wi-fi direct™. Wi-Fi Alliance, available at http://www.wi-fi.org/discover-and-learn/wi-fi-direct, 2013.

[App13] AppNearMe. Nfc applications framework. AppNearMe, available at http://www.appnearme.com/, 2013.

[ASR99] E. Rescorla A. Schiffman and Inc. RTFM. The secure hypertext transfer protocol. RFC 2660, available at http://tools.ietf.org/html/rfc2660, August 1999.

[Bib05] Erin Biba. Does your wi-fi hotspot have an evil twin? PCWorld, available at http://www.pcworld.com/article/120054/article.html, 2005.

[Che13] Jesse Chen. Introducing instawifi. InstaWifi, available at http://www.instawifi.jessechen.net/, 2013.

[Cla09] Sarah Clark. Telecom italia and atm to launch nfc ticketing service in milan. NFC World, available at http://www.nfcworld.com/2009/04/24/3972/telecom-italia-and-atm-to-launch-nfc-ticketing-service-in-milan/, 2009.

[Cla12] Sarah Clark. Telecom new zealand and westpac test nfc with auckland transport. NFC World, available at http://www.nfcworld.com/2012/04/30/315380/

telecom-new-zealand-and-westpac-test-nfc-with-auckland-transport/, 2012.

[Cru11] André Cruz. NFC and mobile payments today. 2011.

[Dav12] Joe Davies. A definitive list of nfc phones. NFC World, available at http://www.nfcworld.com/nfc-phones-list/, 2012.

[DL12] B Dodson and M S Lam. Micro-interactions with NFC-enabled mobile phones, 2012.

[EA12] Techin Eamrurksiri and Xiang Ang Qi. Near Field Communication. Technical Report May, January 2012.

[Ele06] Electronic Engineering Times Asia. NFC delivers intuitive , connected consumer experience. 2006.

[Fis09] J Fischer. NFC in cell phones: The new paradigm for an interactive world. *IEEE Communications Magazine*, 47(6):22–28, 2009.

[For06] NFC Forum. Nfc data exchange format (ndef). Technical Report July, 2006.

[For07] The Broadband Forum. TR-069 - CPE WAN Management Protocol. Technical Report July, 2007.

[For08] The Broadband Forum. Oma device management protocol. Technical Report Jun, 2008.

[For13] NFC Forum. The nfc forum. NFC Forum, available at http://www.nfc-forum.org/aboutus/, 2013.

[Gal11] Francesco Gallo. NFC Tags - A technical introduction, applications and products. Technical Report December, 2011.

[GMS08] J. Langer G. Madlmayr and J. Scharinger. Results From a NFC User Experience Trial. 2:7, February 2008.

[Goo12] Google. We're building a smarter wallet. Google, available at http://www.google.com/wallet/, 2012.

[Goo13a] Google. Android technical information. Android Open Source Project, available at http://source.android.com/tech/index.html, 2013.

[Goo13b] Google. Android, the world's most popular mobile platform. Android Developers, available at http://developer.android.com/about/index.html, 2013.

[Goo13c] Google. Near field communication. Google, available at http://developer.android.com/guide/topics/connectivity/nfc/index.html, 2013.

[Goo13d] Google. Settings. Android Developers, available at http://developer.android.com/reference/android/provider/Settings.html, 2013.

[Hat11] Grant Hatchimonji. 4g world: The future of nfc. Brighthand, available at http://www.brighthand.com/default.asp?newsID=18321&news=near+field+communications+nfc+4G+world, 2011.

[HB] Ernst Haselsteiner and Klemens Breitfuß. Security in Near Field Communication (NFC) - Strengths and Weaknesses. Technical report.

[Hop09] Bruce Hopkins. Faster data transfer with bluetooth and contactless communication. Oracle, available at http://www.oracle.com/technetwork/articles/javame/nfc-bluetooth-142337.html, 2009.

[Inf97] Infrared Data Association. Serial infrared physical layer link specification. Technical report, November 1997.

[Int08] ECMA International. Near field communication - interface and protocol (nfcip-1). Technical Report February, 2008.

[Int10] ECMA International. Near field communication - interface and protocol (nfcip-2). Technical Report June, 2010.

[Ker11] Martin Kerschberger. Near Field Communication - A survey of safety and security measures. 2011.

[KMl10] Henning Siitonen Kortvedt and Stig F Mjø lsnes. Eavesdropping Near Field Communication. pages 57–68, 2010.

[Kum10] Anurag Kumar. *Near Field Communication - a seminar report*. PhD thesis, 2010.

[Mas12] MasterCard. Tap to pay. MasterCard, available at http://www.mastercard.us/paypass.html#/home/, 2012.

[MKH+12] J Morak, H Kumpusch, D Hayn, R Modre-Osprian, and G Schreier. Design and evaluation of a telemonitoring concept based on NFC-enabled mobile phones and sensor devices. *IEEE Transactions on Information Technology in Biomedicine*, 16(1):17–23, 2012.

[MLK+09] G Madlmayr, J Langer, C Kantner, J Scharinger, and I Schaumuller-Bichl. Risk Analysis of Over-the-Air Transactions in an NFC Ecosystem, 2009.

[MP11] S Miranda and N Pastorelly. NFC mobiquitous information service prototyping at the university of nice sophia antipolis and multi-mode NFC application proposal. In *Proceedings - 3rd International Workshop on Near Field Communication, NFC 2011*, 3rd International Workshop on Near Field Communication, NFC 2011, pages 3–8, I3S (Kewi Group), University of Nice Sophia Antipolis, Greece, 2011.

[MRT12] A Matos, D Romao, and P Trezentos. Secure hotspot authentication through a Near Field Communication side-channel. pages 807–814, 2012.

[Nat01] National Institute of Standards and Technology. Fips 197: Advanced encryp- tion standard (aes). 2001.

[NFC06] NFC Forum. Nfc data exchange format (ndef) - technical specification. Technical Report Jul, 2006.

[NFC11a] NFC Forum. Logical link control protocol - technical specification. Technical Report Jun, 2011.

[NFC11b] NFC Forum. Simple ndef exchange protocol - technical specification. Technical Report Aug, 2011.

REFERENCES

[NXP07] NXP Semiconductors. Pn532 user manual. Technical Report Nov, 2007.

[PKH+06] S Patel, J Kientz, G Hayes, Sooraj Bhat, and G Abowd. Farther than you may think: An empirical investigation of the proximity of users to their mobile phones. *UbiComp 2006: Ubiquitous Computing*, pages 123–140, 2006.

[Rai13] RailsGuides. Getting started with rails. Rails, available at http://guides.rubyonrails.org/getting_started.html, 2013.

[RSA12] RSA Laboratories. Pkcs #1 v2.2: Rsa cryptography standard. October 2012.

[SIG13] Bluetooth SIG. Fast facts. Bluetooth SIG, available at http://www.bluetooth.com/Pages/Fast-Facts.aspx, 2013.

[Tag13a] Tagstand. Nfc task launcher. Tagstand, available at http://launcher.tagstand.com/, 2013.

[Tag13b] Tagstand. Nfc task launcher. Google Play, available at https://play.google.com/store/apps/details?id=com.jwsoft.nfcactionlauncher, 2013.

[Tim07] NFC Times. Austria: 'rollout' uses nfc reader mode to sell tickets and snacks. NFC Times, available at http://nfctimes.com/project/austria-rollout-uses-nfc-reader-mode-sell-tickets-and-snacks, 2007.

[Tim10] NFC Times. Germany: Transit officials enable users to tap or scan in new trial. NFC Times, available at http://nfctimes.com/project/germany-transit-officials-enable-users-tap-and-scan-new-trial, 2010.

[Uni13] International Telecommunications Union. The world in 2013: Ict facts and figures. International Telecommunications Union (ITU), available at http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2013.pdf, 2013.

[Upa12] Nikita Upadhyay. Telcos tap new tech to drive m-commerce. The Financial Express, available at http://www.financialexpress.com/news/telcos-tap-new-tech-to-drive-mcommerce/951886/0, May 2012.

[VK11] R Verdult and F Kooman. Practical attacks on NFC enabled cell phones. In *Proceedings - 3rd International Workshop on Near Field Communication, NFC 2011*, 3rd International Workshop on Near Field Communication, NFC 2011, pages 77–82, Institute for Computing and Information Sciences, Radboud University Nijmegen, Netherlands, 2011.

[Wag08] S Wagner. Zero-Configuration of Pervasive Healthcare Sensor Networks, 2008.

[Web13] Webopedia. Wi-fi. Webopedia, available at http://www.webopedia.com/TERM/W/Wi_Fi.html, 2013.

[Wik13] Wikipedia. List of nfc-enabled mobile devices. Wikipedia, available at http://en.wikipedia.org/wiki/List_of_NFC-enabled_mobile_devices, 2013.